Clustering Paths With Dynamic Time Warping

Rainer Koschke
University of Bremen, Germany
orcid.org/0000-0003-4094-3444

Marcel Steinbeck
University of Bremen, Germany
marcel@informatik.uni-bremen.de

***Abstract*—Studying software visualization often includes the evaluation of paths collected from participants of a study (e.g., eye tracking or movements in virtual worlds). In this paper, we explore clustering techniques to automate the process of grouping similar paths. The heart of the evaluated approach is a distance metric between paths that is based on dynamic time warping (DTW). DTW aligns two paths based on any given distance metric between their data points so as to minimize the distance between those paths—alignment may stretch or compress time for best fit. With a data set of 127 paths of professional software developers exploring code cities in virtual reality, we evaluate the clustering based on objective quality indices and manual inspection.**

***Index Terms*—dynamic time warping, clustering, path data, time series**

I. Introduction

We are studying human movements and navigation behavior in virtual software cities [1], where we track the positions visited by participants as paths in 3D space. In the set of paths, we then look for similar behavior. However, if there are many paths, we would like to cluster them into groups of similar paths to ease our manual inspection. A similar need for clustering paths occurs also in eye-tracking studies where, for instance, humans are tracked to find how they view at software visualizations or studies on how developers read code. In this paper, we explore how to leverage clustering in order to group similar paths automatically. Clustering requires a distance function between paths, e.g., the Euclidean distance. Because our paths (movement paths) may vary in length and also in timing—some humans move faster than others—, we are exploring whether dynamic time warping (DTW) can be used as an alignment technique of paths for computing the Euclidean distance. DTW is capable of stretching or compressing time so that the distance between paths is minimized, thus, abstracting from different lengths and speeds. We address the following research questions:

(RQ1) Does automated clustering support us in investigating similar movement behaviour?

(RQ2) Is DTW a suitable technique to determine the distances between paths?

(RQ3) Can classic clustering validation indices give us hints on "good" clusters?

Section II presents related research. Section III delves into clustering paths based on DTW and clustering validation indices proposed to assess the goodness of clusters. Section IV reports on our study in which we applied those concepts to paths collected from 127 professional developers moving in a virtual code city. Section V, finally, concludes.

II. Related Research

Clustering as such is a long established field of research. We will cite the relevant general works later in Section III. In software engineering, clustering has been used, too, in particular for architecture recovery where similar modules are clustered [2]. Clustering of time series based on dynamic time warping (DTW), however, is a very rare topic in software engineering. Despite of extensive search, we found only very few papers: Bouktif et al. proposed an approach based on DTW to detect change patterns in CVS repositories in order to answer the question which other files in a software system should be changed if a particular file was modified [3]. The authors noted that precision and recall of their approach could be improved by using clustering techniques in addition to DTW. Ding et al. have conducted a study on the state-of-the-art representation methods and similarity measures for time series data [4], among which was also DTW, based on the *UCR Time Series Data Mining Archive* [5]. This archive of time series, however, has no data specific to software engineering aspects, in particular no data related to movement paths. Abdelkader and Mimoun have proposed to model code sequences of functions as time series and then to find cloned functions via DTW [6]. Our need for clustering paths arose from our study of human movements and their navigation behavior in virtual software cities [1], a topic that occurs and is addressed also in other domains. For instance, Amornbunchornvej and Berger-Wolf proposed a framework to infer following strategies of social groups based on time series of movement data [7]; they did not, however, use DTW. We are not aware of any work in software engineering that clusters paths based on DTW.

III. Clustering Paths

This section describes how to cluster paths based on dynamic time warping.

A. Path Distances and Dynamic Time Warping

Let $Space^n$ be an n-dimensional Cartesian coordinate system. Every *position* $p \in Space^n$ can be represented as a vector $p = (x_1, x_2, \ldots, x_n)$. In case of eye-tracking or mouse movements on a two-dimensional display, n=2 and $Space^2$ represents the set of pixels of the display. In our case of paths in virtual reality spaces, n=3 and $Space^3$ is the set of positions a player can reach in the virtual world. Without loss of generality, we follow the convention by Unity—the game engine we are using to render our virtual worlds—where every position is a vector (x_1, x_2, x_3) as follows: x_1 is the direction

978-1-7281-9914-6/20/$31.00 ©2020 IEEE
DOI 10.1109/VISSOFT51673.2020.00014

towards right, x_2 is the direction up, and x_3 is the direction forward.

A *path* is an ordered sequence of positions $[p_1, p_2, \ldots, p_m]$ where $m \geq 0$ is its *length*. A *timed path* is a path for which a function t:*Space*$^n \rightarrow \mathbb{R}_0^+$ exists that assigns a point in time (a non-negative real number) to every position in that path describing when this position was reached. We assume that $t(p_1)=0$, that is, every path starts at time zero. Every pair (p_i, p_{i+1}) in a path denotes a movement from one position p_i to the immediate next p_{i+1} where $t(p_i)<t(p_{i+1})$. In our recordings, we used a fixed resolution of time, that is, $t(p_{i+1})-t(p_i)=$*period* for all neighboring positions of all paths where the constant *period* was 0.5 seconds. Anyhow, whether the positions were recorded at a fixed interval or even uniformely across paths does not really matter in the following. As a matter of fact, the timing is neglected by dynamic time warping altogether—only the order counts, but it can be used to further study the differences between similar paths.

Given the above properties of paths, a path forms a *time series*, which is defined as a series of data points indexed in time order and, thus, is amenable to time-series analysis. In particular, we can measure the similarity between two paths using a distance metric between positions contained in those paths. The most simple case for such a similarity measure would be if both paths had equal lengths and their corresponding positions where recorded at the same relative points in time. Then both paths would form a vector of the same length and a distance metric, such as Euclidean distance, could be applied to every pair of corresponding positions of the two paths. Yet, if the paths have different lengths and/or the speed of movements differs between paths (so that positions are reached at different points in time) but one wishes to abstract from the speed—as it is the case for us—, it is not so simple. Here, *dynamic time warping (DTW)* can help.

The basic idea behind DTW is to stretch or compress two given time series locally in order to make one resemble the other as much as possible. In our case, DTW abstracts from the timing but not from the order in which positions of different paths were reached. Based on the resulting alignment, the distance between two paths is computed by summing up the distances of the individual (aligned) positions. DTW algorithms have been popularized in the seventies of the last century in the context of speech recognition to account for differences in speaking rates between speakers and/or utterances [8], [9].

DTW allows us to compare a time series (or timed path) $P = (p_1, \ldots, p_N)$—called the *test* or *query*—against a *reference* time series (timed path) $Q = (q_1, \ldots, q_M)$ [10]. For clarity, we are using the symbol $i \in [1, N]$ as an index for positions in P and $j \in [1, M]$ for those in Q. Every p_i as well as every q_j is an element of *Space*^{n} and a difference can be measured between the two, for instance, as the Euclidean distance. Which distance to use in DTW can be decided freely. For generality, we assume that we have a non-negative *local distance function* f defined between any pair of positions p_i and q_j. We will use the shortcut $d(i, j) = f(p_i, q_j) \geq 0$ in the following. In our application of DTW, we use the Euclidean distance, which comes quite naturally for three-dimensional spaces. Yet, other distances could be used, too.

The central concept of DTW is the alignment of two time series enabling the calculation of the distance between corresponding positions of the two time series, called the *warping curve* $\phi(k) = (\phi_P(k), \phi_Q(k))$ with $k \in [1, T]$ where T=*max*(N,M) and $\phi_P(k) \in [1, N]$ is a position in P aligned with a position $\phi_Q(k) \in [1, M]$ in Q. The purpose of the two *warping functions* $\phi_P(k)$ and $\phi_Q(k)$ is to remap the indices of P and Q, respectively. Given ϕ, the *average accumulated distortion* $d_\phi(P, Q)$ between the warped time series P and Q can be calculated as follows:

$$d_\phi(P, Q) = \sum_{k=1}^{T} d(\phi_P(k), \phi_Q(k)) \cdot m_\phi(k)/M_\phi \qquad (1)$$

where $m_\phi(k)$ is a per-step weighting coefficient and M_ϕ is the corresponding normalization constant used to ensure that the accumulated distortions are comparable along different alignment paths

Because the alignment should not be arbitrary, constraints may be imposed on ϕ. For instance, *monotonicity* is generally requested to preserve the time ordering and to avoid meaningless loops: $\phi_P(k+1) \geq \phi_P(k)$ and $\phi_Q(k+1) \geq \phi_Q(k)$. Another frequent constraint is that the starts of both time series must be the same, that is, $\phi_P(1) = \phi_Q(1) = 1$. This constraint is fulfilled in our case because all our paths start at the same location. In eye-tracking studies, however, that is generally impossible to enforce. Frequently, it is also requested that the ends of the two time series must match: $\phi_P(T)=N$ and $\phi_Q(T)=M$. This constraint is not fulfilled in our application of DTW because our participants were allowed to stop at any time. If the start and end of both time series are matching, the two time series are said to be *globally aligned*.

Even in the presence of the above constraints, there are still many degrees of freedom. These degrees of freedom allow a multitude of possible instances of ϕ to align two time series. What we search for is the optimal alignment, keeping in mind that we use it to calculate the distance between the two times series. An optimal ϕ is, thus, one that minimizes the distance, that is, a ϕ such that $D(P, Q) = \mathit{mind}_\phi(P, Q)$. Algorithms to find optimal ϕs (there may, of course, be mutliple ones) are based on dynamic programming and run in $O(N \cdot M)$ time [11]. For the study presented in this paper, we used the R package *dtw* [12] to find an optimal ϕ and to calculate the difference between aligned time series (see Section IV-C).

The algorithms allow one to specify further constraints beyond monotonicity or global alignment, in particular different notions of continuity. The *symmetric continuity constraint*, for instance, implies that arbitrary time compressions and expansions are allowed and that all elements must be matched: $|\phi_P(k+1)-\phi_P(k)| \leq 1$ and $|\phi_Q(k+1)-\phi_Q(k)| \leq 1$. Similar forms of constraints are possible, too. While the symmetric continuity constraint disallows skipping positions entirely, one could as well want to just limit the number of consecutive positions being skipped (i.e., those left unmatched) by an

upper limit greater than 0. Alignments in general are obtained by duplicating positions, that is, one lets a single position in P match multiple (consecutive) positions in Q, or vice versa. The number of repeated positions that can be matched consecutively or skipped, respectively, puts limits on the local slope of the warping curve. The *warping curve* can best be imagined as the line in a two dimensional chart, where one axis represents P and the other axis represents Q. There is a dot in this chart if the respective positions of P and Q match. That is, the warping curve pictures $\phi(k)$=($\phi_P(k)$,$\phi_Q(k)$) for all values of k. If P and Q would be of equal length and all their positions matched perfectly, the warping curve would be the diagonal. If some positions were instead to be skipped to obtain an alignment, the slope of the warping curve would defer from 1. If one wants to put a limit on how many positions can be skipped, one can simply restrict the slope of the warping curve. This is achieved by so called *step patterns*, which can be passed as an argument to a DTW algorithm. A step pattern lists sets of allowed transitions between matched pairs and the corresponding weights. In other words, a step patterns defines the admissible value of $\phi(k+1)$ given $\phi(k)$,$\phi(k-1)$, etc. The associated weight ($m_\phi(k)$ in Equation (1)) is a penalty for selecting a particular continuation of $\phi(k)$. The Levenshtein distance or Smith-Waterman's distance to align two sequences have a similar concept of penalty, but in DTW there is no additive penalty for duplicating or skipping elements. That is, DTW is better suited when time may be arbitrarily compressed or expanded than these other measures for aligning sequences.

We are using the simple step pattern named *symmetric2*, defined as follows (given two indices, i and j, for ϕ_P and ϕ_Q, respectively):

$$c[i,j]=min(c[i-1,j-1]+2\cdot d(i,j),c[i,j-1]+d(i,j),c[i-1,j]+d(i,j))$$

where c is the cost matrix filled by the dynamic programming algorithm to find an optimal ϕ (each cell contains the alignment costs for a particular path in the warping curve) and $d(i,j)$ is the distance as defined above. Continuing the warping curve to the right in the alignment matrix is as costly as continuing it upward for *symmetric2*. Taking a diagonal path has the same cost as the sum of the costs of going up and right. This offers maximal flexibility in aligning the next position based on the immediate previous ones in the alignment. It means that we can arbitrarily compress or expand time to find the optimal alignment. Moreover, it is symmetric so that the resulting distance metric is symmetric, too. This has computational advantages because only half of the distance matrix for the subsequent clustering must be computed and also advantages for cluster evaluation (see Section III-C), but more importantly there is no reason for us to believe that a distance from P to Q should be anything else but the distance from Q to P when comparing paths. For these reasons, this choice seems justified for our application of DTW and it is also among the most popular step patterns in other applications of DTW. Yet we note that there are many alternatives proposed in the literature [10], [13], [14].

B. Clustering

The *average accumulated distortion* $d_\phi(P,Q)$ between the warped time series P and Q defined by Equation 1 through DTW gives us a measure of the similarity of two paths viewed as time series where time can be compressed and expanded. A distance of zero means that two subjects visited the very same locations in the same order, although at possibly different speeds. This distance measure allows us to cluster similar paths to find similar behavior of subjects.

The process of clustering paths can be automated by classical clustering algorithms. Cluster analysis is an unsupervised classification, that is, it does not require any oracle that gives hints on correct or false clusters. There is a plethora of different kinds of clustering algorithms. Classical types of clusterings for time series are hierarchical, partitional, and fuzzy clustering. Fuzzy clustering yields a likelihood that an element belongs to a cluster, while hierarchical and partitional clustering both yield crisp clusters. Although it is straightforward to change a fuzzy clustering into a crisp clustering by simply assigning an element to the cluster for which it has the greatest likelihood, it adds yet another parameter whose influence needs to be investigated. That is why we will look into fuzzy clustering only in future research.

All the different types of clustering require a measure of distance (or dually, similarity) between the elements to be clustered. In our case, $d_\phi(P,Q)$ is used for that. Beyond that they have different additional requirements and distinct methods we briefly summarize in the following.

Hierarchical clustering is a bottom-up approach that starts with singleton clusters—one for each element to be clustered—and then successively merges clusters with minimal distance (or, dually, highest similarity) until a cluster is reached containing all elements. When clusters contain more than one element, the initial distance metric for single elements is no longer directly applicable because it is defined in terms of two elements and not two groups of elements. Consequently, a second measure of distance is required that must be defined for groups of elements to decide whether two groups that are not both just singletons should be merged. Here again many alternatives exist. The classic measures for inter-group distance are *single linkage*, where the inter-group distance is that of the closest (least dissimilar) pair, *complete linkage*, where the farthest (most dissimilar) pair is chosen for the inter-group distance, or *unweighted average linkage*, where the average distance between all pairs of elements of the two clusters to be merged serves as the inter-group distance. *Single linkage* frequently exhibits *chaining*, which is the tendency to incorporate single elements into existing clusters rather than creating new clusters and may create clusters with elements that are far away from each other. Because we want more cohesive clusters of paths, we do not use it. *Complete linkage*, on the other hand, tends to find compact clusters of approximately equal diameters [15]. *Unweighted average linkage* is a compromise between the two. We refer the reader to the literature for other measures of inter-group distances [15].

The merging of clusters by *hierarchical clustering* can be tracked and forms a tree where every inner node represents a merge of two clusters associated with the distance between those. This tree can be visualized as a so-called *dendrogram* (cf. Figure 1), where the x axis lists all elements to be clustered and the y axis depicts the distances at which two clusters were merged. Thus, hierarchical clustering does not create only a single partition of the data. Instead, many can be induced from the dendrogram by selecting a distance threshold at which to cut the tree. Which threshold is suitable can be derived manually by inspecting the dendrogram's possible clusters—which may be tedious and somewhat subjective—or by the means we describe in Section III-C.

Partitional clustering is rather a top-down approach that starts with an existing clustering which is then successively refined into more cohesive clusters until a given number, k, of requested clusters is reached. As opposed to hierarchical clustering, the parameter k must be specified beforehand. There are different kinds of partitional clustering algorithms, but all work in the following common way. First, k centroids are randomly initialized, usually by choosing k elements randomly forming inital singleton clusters. Then the distance between all remaining elements and all centroids is calculated, and each element is assigned to the cluster of its closest centroid. A prototyping function is applied to each cluster to update the corresponding centroid of each cluster after each round of refinement. Then, distances and centroids are updated iteratively until a certain number of iterations has been reached, or no element changes clusters anymore. A *centroid* is intended to characterize all members of the cluster and may not necessarily be a member of the data set. For instance, it could be an average vector over all cluster members.

Very popular partitional algorithms are k-means and k-medoids [16]. *K-means* chooses a time series as the centroid that is calculated as the average of the cluster members along each dimension of the underlying multi-variate data ($Space^n$). More precisely, suppose we have a cluster of s time series $\{ts^1, \ldots, ts^s\}$. Each time series ts^l ($l \in [1, s]$) in the cluster is a sequence $(p^{l,1}, \ldots, p^{l,M})$ of M positions in $Space^n$, thus, each position $p^{l,m}$ ($m \in [1, M]$) is a vector $(x_1^{l,m}, \ldots, x_n^{l,m})$. Then the centroid of k-means is a time series $(p^1, \ldots, p^M)$ where each position p^i ($i \in [1, M]$) is the average vector of all vectors $(x_1^{*,i}, \ldots, x_n^{*,i})$ among the s series that belong to the same cluster for all equal time points. Obviously, this calculation is possible only if all time series have the same length. If there are unmatched positions due to different lengths of the time series, there is no one-to-one correspondence between elements of different time series and, hence, it is not obvious how to calculate their average. Moreover, the result may be a time series that is not actually one that was observed but a result from averaging. If the goal is to find a real observation as a representative of a cluster, the centroid calculated by k-means is not helpful. In our application of DTW, the time series (paths) have very different length. Moreover, we want to study true observations. That is why k-means is not suitable for our context.

K-medoids, also known as *partition around medoids (PAM)*, minimizes the sum of distances between points labeled to be in a cluster and a point designated as the center of that cluster, i.e., the centroid. In contrast to k-means, k-medoids chooses observed time series as centroids (also known as *medoids* or *exemplars*). A *medoid* of a cluster is an element from this cluster whose average distance to all other elements of that cluster is minimal, that is, it is the most centrally located point in the cluster. K-medoids is said to be more robust to noise and outliers as compared to k-means because it minimizes a sum of general pairwise distances instead of mean distances. In addition, its centroid is an element of the original data set, that is, a true observation.

C. Clustering Evaluation

Partitional clustering requires a parameter k determined upfront for the number of requested clusters. Similarly, the dendrogram produced by hierarchical clustering can be cut to obtain k clusters. The remaining question then is what is a suitable value for k to obtain "good" clusters? Hierarchical clustering at least allows us to inspect the dendrogram—which presents not just a single partitioning of the data but many in terms of subtrees that can be cut from the dendrogram: each inner node represents a merge of clusters and is associated with the distance for the merged clusters. This way one could determine a distance threshold at which it becomes doubtful to further merge clusters. In case of partitional clustering, one could generate clusters for a range of different values for k and then inspect the results. In both cases, this manual process is tedious and may also be subjective.

If one wants to fully automate this process and to make it as objective as possible, other approaches may be considered. An idea is to define a quality measure for the resulting clusters—a *cluster validity index (CVI)* in clustering terminology—against which suitable values for k are to be optimized. There are many such CVIs proposed in the clustering literature that can rougly be classified as internal, external or relative depending on how they are computed [17]. *Internal validation* validates a partition by examining only the partitioned data whereas *external validation* compares the partition with a correct partition. The latter, obviously, requires a ground truth, which contradicts the idea of unsupervised classification. Given two partitions (one of which is the ground truth), X and Y, of a data set with N elements, the *Rand Index (RI)* [18] is the number of pairs that are in the same cluster in X and also in the same cluster in Y, *TP*, plus the number of pairs that are in different clusters in X and also in different clusters in Y, *TN*, divided by the total number of combinatorically possible pairs: $(TP+TN)/(N(N-1)/2)$. It is a measure of accuracy with a value range $[0, 1]$ and is sensitive to imbalanced data. The *Adjusted Rand Index (ARI)* puts the Rand Index in relation to a random clustering. The random clustering created through permutation is a baseline that a good clustering should outperform. It can take on negative values if a clustering is worse than the random baseline. The *Jaccard Index (JI)* is similar to the Rand Index but disregards the pairs of elements that are

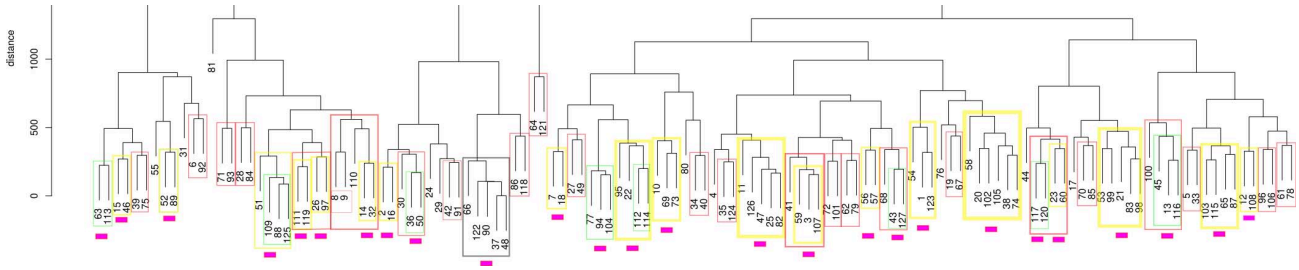


Fig. 1: Annotated dendrogram of the hierarchical clustering of paths with complete linkage.

in different clusters for both clusterings (and consequently not sensitive to imbalanced data): $TP/(TP{+}FN{+}FP)$ where *FP* is the number of pairs that are in the same cluster in X but not in Y and *FN* is the number of pairs that are in the same cluster in Y but not in X. The value of this index falls into the range $[0, 1]$ where 1 is the best value. The *Fowlkes–Mallows Index (FM)* is defined as $\sqrt{TP/(TP{+}FP)\cdot TP/(TP{+}FN)}$ [19], again with a value range $[0, 1]$ with 1 being the optimum. Fowlkes and Mallows observed that the Rand Index approximates 1 for $k \to N$ (where k is the number of clusters and N the number of elements), that is, the Rand Index is suited only for k considerably lower than N. Another external CVI, named *Variation of Information (VI)*, was proposed by Meila [20] that is based on entropy. The precise details of that information-theoretical measure can be found in the original paper; here we just summarize the intuition of what it captures. Informally, its value corresponds to the amount of information that is gained minus the amount of information lost when going from X to Y and vice versa. Its value is not bound by a constant value. The closer the value is to zero, the better.

Relative validation compares two partitions to each other, where often an external or internal CVI is used for this comparison. We will follow this approach by generating clusters for different values of k, gather different internal CVIs for each resulting partition, and then compare those CVIs for partitions obtained from different choices of k to each other. The relation between different values of k and the CVIs can be plotted as a two-dimensional graph. Often such graphs resemble an 'elbow' shape, that is, a CVI decreases monotonically as k increases and from some k onwards the decrease of the curve flattens markedly [21], [22]. The location of the 'elbow' suggests the appropriate value for k then. We could retrieve k by simply looking at the chart. To provide a more objective and automatable selection, the *gap statistic* provides a statistical procedure to formalize this heuristic [23]. The idea of the *gap statistic* is to standardize the graph plotting the relation between k and a CVI by comparing it with its expectation under an appropriate null reference distribution for the CVI. The estimate of the optimal k is then the value of k for which the standardized CVI falls the farthest below the reference curve (the one obtained assuming the null reference distribution). More precisely, let $\mathbb{C}$ be the set of clusters of paths with $k = |\mathbb{C}|$ and $A \in \mathbb{C}$ be a cluster, then $D_A = \sum_{P,Q \in A} d_\phi(P, Q)$ is the sum of the pairwise distances for all paths in cluster A. The pooled within-cluster distances around the cluster mean, W, (the factor 2 in the equation makes this work exactly) is defined as follows:

$$W(\mathbb{C}) = \sum_{A \in \mathbb{C}} \frac{D_A}{2|A|} \tag{2}$$

The *gap statistic* is then defined to be the following comparison between W and an appropriate null reference distribution of the data (see also [24]):

$$gap(\mathbb{C}) = E_n^*\{log(W(\mathbb{C}))\} - log(W(\mathbb{C})) \tag{3}$$

where the values are standardized by the *log* function and E_n^* denotes the expectation under a sample of size n from the reference distribution. The mean and standard deviation of the sample can be determined and the recommended value of k will be the sample mean value maximizing *gap*($\mathbb{C}$) after the sampling distribution was taken into account. As a bonus, the standard deviation allows one to assess the confidence into the chosen value: the lower the standard deviation, the higher the confidence into the selected mean chosen for k. This estimate of k is generally applicable to any clustering method and distance measure. More details in particular about suitable null reference distributions can be found in the original paper [23].

Next we will delve into popular CVIs for internal validation. There is a plethora of CVIs and we cannot explain and investigate all of them. The interested reader is referred to the extensive comparative study of CVIs by Arbelaitz et al. [17] (the paper does not evaluate those CVIs for comparing time series, however). For practical reasons, we limit ourselves to those offered by the R function *cvi*. This function is part of the R package *dtwclust* [25] we use for clustering our time-series leveraging DTW and offers multiple internal CVIs for DTW-based clustering we will explain shortly. Each index has its own range of values. Some of them are to be minimized and some of them to be maximized to get good clusters.

These CVIs can be used to evaluate the result of a clustering algorithm regardless of how the partition came to be. There is no way to know in advance which CVI works best as those may depend upon the specific application. Because we are not aware of any prior study to cluster paths, we will gather and compare all of them in Section IV. If there is no consensus among the CVIs, at least a majority vote might be used to decide on the best value for k.

The *Silhouette index*, *Sil*, by Rousseeuw [26] is a measure for how well objects lie within their cluster and which ones are merely in between clusters. Let P be a path clustered and A be the cluster P belongs to. Then we can calculate

the average distance $a(P)$ between P and every other $Q \in A$ (where $P \neq Q$). The underlying distance measure pairwisely applied to P and all Q is d_ϕ as defined by Equation (1) as both P and Q are paths in our case. Accordingly, $a(P)$ tells us how well P fits into its cluster A it was assigned to by the clustering algorithm. Complementary to that, we can calculate the average distance $d(P, C)$ between P and all elements in another cluster $C \neq A$, which gives us a measure of how well P would fit into C instead. When $d(P, C)$ is computed for all clusters $C \neq A$, we can select the smallest value of function d and denote it by $b(P) = min_{C \neq A} d(P, C)$. The cluster B for which this minimum is attained is the second-best choice for clustering P. The final cluster validation index *Sil* is then defined as the averaged silhouette over all clustered elements as follows ($\mathbb{C}$ is the set of clusters of paths and N is the number of paths in the data set to be clustered):

$$Sil(\mathbb{C}) = \frac{1}{N} \sum_{A \in \mathbb{C}} \sum_{P \in A} \frac{b(P) - a(P)}{max\{a(P), b(P)\}}$$

Sil lies in the range $[-1, 1]$, where a higher value indicates a better clustering. If cluster A contains only P and no other element, $Sil(P)$ is defined to be the neutral value zero.

The *Dunn index*, *D*, [27] is a ratio-type index (i.e., yields a value between 0 and 1, where 1 is the best) where the cohesion of a cluster is estimated by the nearest neighbour distance and the separation by the maximum cluster diameter. The index is defined as follows:

$$D(\mathbb{C}) = \frac{min_{A \in \mathbb{C}}\{min_{B \in \mathbb{C}-A}\{d(A, B)\}\}}{max_{A \in \mathbb{C}}\{\Delta(A)\}}$$

where $d(A, B) = min_{P \in A}\{min_{Q \in B}\{d_\phi(P, Q)\}\}$ and $\Delta(A) = max_{P \in A, Q \in A}\{d_\phi(P, Q)\}$.

The *COP* index is a ratio-type index, too, where the cohesion is estimated by the distance from the elements in a cluster to the centroid of their cluster and the separation is based on the furthest neighbour distance [28]. Its definition is as follows (the lower the value, the better; C_A denotes the centroid of a cluster A; N is the total number of paths to be clustered):

$$COP(\mathbb{C}) = \frac{1}{N} \sum_{A \in \mathbb{C}} |A| \frac{1/|A| \sum_{P \in A} d_\phi(P, C_A)}{min_{P \notin A}\{max_{Q \in A} d_\phi(P, Q)\}}$$

The *Davies-Bouldin index*, *DB*, estimates the cohesion based on the distance from the elements in a cluster to its centroid and the separation based on the distance between centroids of different clusters [29]. It is among the most frequently used indices in CVI comparison studies and defined as follows (again, C_X denotes the centroid of a cluster X; the lower the value, the better; $k = |\mathbb{C}|$ is the number of clusters):

$$DB(\mathbb{C}) = \frac{1}{k} \sum_{A \in \mathbb{C}} max_{B \in \mathbb{C}-A}\{\frac{S(A) + S(B)}{d_\phi(C_A, C_B)}\}$$

where $S(A) = 1/|A| \sum_{P \in A} d_\phi(P, C_A)$

The modified *Davies-Bouldin* index*, *DB**, is a variation of *DB* where the minimal distances between cluster centroids are used in the denominator rather than maxima and defined as follows (again, the lower, the better):

$$DB^*(\mathbb{C}) = \frac{1}{k} \sum_{A \in \mathbb{C}} \frac{max_{B \in \mathbb{C}-A}\{S(A) + S(B)\}}{min_{B \in \mathbb{C}-A}\{d_\phi(C_A, C_B)\}}$$

While *DB* maximizes the fraction of inter- and intra-centroid distances for selected pairs of clusters, *DB** puts the maximal intra-cluster cohesion of pairs of clusters in relation to the minimal inter-cluster centroid distance (the lowest coupling between clusters so to speak), where the pairs considered in the nominator and denominator are completely independent. A rationale for this variation and a comparison between the variation and its original as well as other CVIs can be found in the original paper by Kim and Ramakrishna [30].

The *Calinski-Harabasz index*, *CH*, is a ratio-type index estimating the cohesion of clusters based on the distances from the elements in a cluster to its centroid [31]. The separation, on the other hand, is based on the distance from the centroids to the global centroid. The global centroid is the centroid of the whole data set, that is, it is independent of any clustering. The notion of centroid, however, may depend upon the particular kind of clustering applied. For partitional clustering, we used *partition around medoids (PAM)*, where medoids are used as centroids. As explained in Section III-B, the *medoid* is an element from the data set whose average distance to all other elements of that data set is minimal, that is, it is the most centrally located point in the data set. To be able to make meaningful comparisons between partitional and hierarchical clustering, we will use the mediods for the CVIs gathered for clusters resulting from hierarchical clustering, too. *CH* can be defined as follows (the larger, the better; M is the global medoid):

$$CH(\mathbb{C}) = \frac{(N - k) \sum_{A \in \mathbb{C}} |A| d_\phi(C_A, M)}{(k - 1) \sum_{A \in \mathbb{C}} \sum_{P \in A} d_\phi(P, C_A)}$$

IV. Study

In this section, we describe our study in which we applied the process outlined in the previous section to cluster similar paths. The goals of the study were to investigate the feasibility, practicality, and usefulness of the proposed method. In particular, we wanted to investigate (RQ1) whether we get meaningful groups of similar paths with automated clustering, (RQ2) whether using a DTW-based distance for this clustering is suitable and computationally feasible, and (RQ3) whether classic measures for cluster validation are helpful to further automate the process. All data and R code for our study are made available so that other researchers can replicate and extend our study [32].

A. Visualization Environment

The paths we analyze in this study represent movements of developers in a virtual world visualizing software based on the software-as-a-city metaphor as created by our visualization tool *SEE* (Software Engineering Experience). The scale of the city was chosen relative to the height of humans, that is, the city buildings appeared in realistic height relative to the player's height so that a player would be able to move

through the streets in an ego perspective. A player steered himself or herself through the virtual world in a continuous motion wearing a head-mounted display and using hand-held VR controllers freely in all three axes (with full-room tracking mode enabled on an area of around 2×2 meters). It was even possible to move under the city.

The underlying data model visualized by SEE is a hierarchical graph, that is, nodes can be nested forming a forest of inner and leaf nodes. The visual components of nodes and edges can be freely configured in SEE. The mapping used in the study is as follows (see Figure 2). Leaf nodes of the underlying dependency graph represent source-code files and are mapped to three-dimensional blocks where the width, height, and depth of the blocks encode a certain metric (width: lines of code, height: fraction of duplicated code, depth: number of tokens; color gradient: McCabe complexity). Inner nodes represent directories containing files or other directories. The hierarchical structure of the nodes is depicted by recursively composing blocks into visual segments. How these segments are determined and where to place visual elements in sections is left to a layout algorithm (see below). Edges of the underlying dependency graph can represent any type of binary relation among software elements. In our study, edges connect two blocks (source-code files) if they share duplicated code. They are visualized by means of hierarchically bundles splines [33], connecting the corresponding blocks. In general, edges are directed, which is represented in SEE through a color gradient. For duplicated code, the direction usually does not matter, however.

The underlying dependency graph visualized in our study is for a real system. We analyzed duplicated code in the network subsystem of the Linux kernel. The resulting graph consists of 1,464 nodes (each representing a distinct source code file or directory) and 1,749 edges (each representing a code fragment shared between two files). The dependency graph including all size-related metrics was extracted by a static analysis offered by tools of the Axivion suite [34]. Based on this graph, we generated a single scene (a single *world*) in which analysis results are visualized using four different layouts (*Circular Balloon*, *Circle Packing*, *Treemap*, and *EvoStreets*; cf. Figure 2). Exactly the same dependency graph and visual mapping was used for all four layouts. The only difference was the layout. Each laid out graph was presented on its own plane clearly separated from the other layouts. Users could freely move within and between the city variants to compare the different layouts with each other.

B. Gathering Path Data

The movement data captured as paths and analyzed in this study is a by-product of a tool presentation held by our research group in co-operation with Axivion [34] at the *Embedded Software Engineering* congress in December 2019 [35]. This congress is one of the largest congresses in the field of embedded software engineering in Germany, hosted every year in Sindelfingen. It is mainly visited by professional C and C++ developers and so we took advantage of the opportunity to present the current state of our visualization tool SEE to a broader audience of professional software developers and software architects. For our live demo, we analyzed the network subsystem of the Linux kernel because it is well known by most of the congress's visitors.

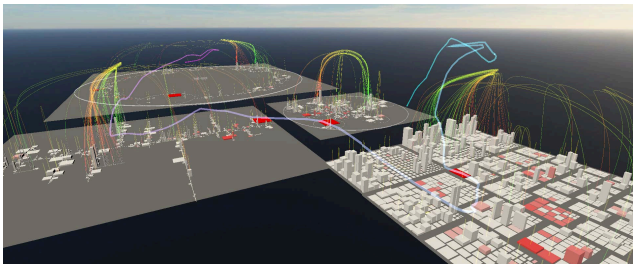


Fig. 2: Virtual code cities used in the study with an example path of one of the participants—the direction of movement is depicted with a color gradient from magenta (start position) to cyan (end position).

The participants got a short introduction in what they would see and how they could move. The interaction was very simple. The direction of the movement in the virtual world was simply the direction in which the controller was pointing. To actually move, the player had to press the trigger of the controller, working as a throttle: the deeper it was pressed, the faster the player would move. All participants understood the interaction within a minute.

The participants did not get any task. They were invited to move around as they wish and just explore what they would find interesting. They could finish whenever they wanted. After all, this was a demo and we did not want to ruin the game experience through any kind of strict instructions. Moreover, we were curious to see what would catch their attention. This unrestricted moving has also advantages for the purpose of our study presented in this paper as it is expected to yield the most heterogeneous paths that are yet not completely arbitrary because certainly the participants were lead by what they saw and found interesting. It is expected that other participants would share their interests and follow similar paths.

After the short introduction, the game was started and all movements were recorded by SEE by capturing the position and rotation of the player in Unity's world space periodically at every 0.5 seconds. The starting point in the virtual world was identical for all participants. The recorded data were kept anonymous and are available [32].

Our data set contains 127 paths. Every path is encoded as a sequence of three-dimensional positions with a time stamp (seconds after game start). Based on these sequences, we calculated the metrics *path length*—sum of the distances between successive positions—and *duration of usage* (i.e., how long our tool was used by the participants)—time between start and end. Our participants, on average, stayed for about five minutes (avg: 5.1, med: 4.6) in virtual reality and traveled a distance of about nine kilometers (avg: 9.7, med: 8.3) therein (one Unity unit corresponds to one meter in real world). For the correlation coefficients *Pearson*, *Kendall*, and *Spearman*,

TABLE I: Elapsed CPU time in seconds.

step	CPU time
initial distance matrix	19.8
partitional clustering p	3.3
hierarchical clustering with complete linkage h_c	1,973.1
hierarchical clustering with average linkage h_a	1,999.9
CVIs for p	832.7
CVIs for h_c	938.7
CVIs for h_a	964.6
gap statistics for p	19.9
gap statistics for h_c	20.0
gap statistics for h_a	19.6

we found moderate correlations between path lengths and duration of usage (all statistically significant with $p \ll 0.01$). Thus, the duration cannot only be explained by the length of a path, in other words, the speed of the players varied, which further suggests the need for an alignment of paths by DTW.

C. Implementation and Run-Time Costs

The exact R packages we used for our implementation can be found in our R script that we made publicly available [32]. All run-time data reported for this study were gathered on a server PC with two Intel(R) Xeon(R) CPUs E5-2690 v4 (14 cores in total) running at 2.60GHz with 255 GB of RAM operating with Ubuntu 16.04.6 LTS. We took advantage of the R package *doParallel* and used all available CPUs. The results are shown in Table I for each major step. The first entry is the initial calculation of the distance matrix for all pairs of paths including the alignment with DTW. This distance matrix is the same for all clusterings and, hence, was re-used for all of them. Every other step—partitional and hierarchical clusterings and gathering of the CVIs and the gap statistic—was run for the whole range of $k \in [2, N-1]$.

As Table I shows, h_c and h_a are expensive operations—by three orders of magnitude more expensive than p. The actual clustering is not the cause of this discrepancy as the trace output of the calls to hierarchical clustering suggests. The clustering is actually finished very quickly. The costs occur for the subsequent extraction of the centroids by hierarchical clustering. An implementation detail seems to be the reason for that, causing the implementation to re-compute the distance matrix again for the centroids even though it was already pre-computed. For partitional clustering, there is an option to let the algorithm for gathering the centroids know that the distance matrix is pre-computed, while there is no such option for hierarchical clustering so that it may be re-used only for the clustering as such but not for the centroids. Likely, this performance issue could be overcome.

Calculating the CVIs is a costly operation for all types of clusterings, whereas the gap statistics is rather efficient to compute. Overall, even though the costs may not allow an interactive exploration, they are still tractable—in particular, if we consider the fact that we are exploring the most extreme value range of $k \in [2, N-1]$.

D. Manual Clustering Validation

The dendrogram produced by hierarchical clustering allows one to start manual the inspection for similar paths at the leaves of the dendrogram and then to move up until a point is reached where clusters are merged that are not similar enough. In this section, we are applying this manual validation process to our data set to answer our research questions RQ1 and RQ2.

We chose hierarchical clustering instead of partitional clustering because we did not know the number of expected clusters in advance, which is a prerequisite for partitional clustering. We looked at the results of complete linkage rather than average clusters because this was expected to give us clusters of paths with mutually shorter distances.

The first author of this paper visualized the paths reported as similar by the dendrogram shown in Figure 1 in an interactive 3D plot where one could look at the paths from all angles. Based on his intuition, he grouped the paths of a cluster into the following categories: (1) similar, (2) somewhat similar (still worth to be looked at to study their differences), (3) not similar, and a special category (4) similar, yet just because the paths are too short. When the paths were recorded, sometimes a recording was started by mistake or was just a test run. There was also one case in which a participant felt umcomfortable and stopped immediately. We could have filtered those paths based on their duration and distance covered, but we left them in the data set to be able to investigate noise in the data and to see whether clustering would be able to detect it, too. The decisions can be found as annotations in Figure 1 as colored rectangles. Category (4) is colored black and one can see that clustering indeed detected those invalid paths. Clusters in category (1) are colored green, (2) in yellow, and (3) in red. When an inner node of the dendrogram was reached at which a cluster was classified in (3), the bottom-up walk of the dendrogram was terminated and re-started at the next not yet visited leaves. The second author investigated the decisions of the first author. When he questioned a decision, both authors discussed the case until a consensus was reached. The reasons for the decisions are documented and publicly available for inspection by other researchers [32].

As one can see in Figure 1, the lower the nodes in the dendrogram, the higher the chances they are valid clusters and vice versa, which suggests that the DTW-based distance between paths matches human intution of proximity for paths in most cases. However, one must also note that the dendrogram does not suggest a consistent constant threshold of acceptable distance alike for all clusters. For instance, the cluster $\{77, 94, 104\}$ was consider similar—category (1)—whereas the cluster $\{2, 16\}$ was classified as only somewhat similar—category (2)—even though their distances are almost the same. The reason for that is that one path of the latter cluster can be considered a continuation of the other path, but their distances are substantially different, whereas the lengths of the former cluster of the three paths are more comparable. The greater distances for the three paths of that clusters is due to different heights in large portions of the paths, while the movement in the x/z plane (in Unity's co-ordinate system) is very similar. That is, the human judges gave more weight to the movements within the plane than in heights, whereas in Euclidean space all axes are treated alike. In other words,

the DTW-based distance covers most of the human intuition of proximity but not in all aspects, e.g., with respect to the subsumption of paths (DTW can compress the end of one path to align it with another path with little penalty) and the semantic weight of the altitude.

E. Cluster Validation Indices

The human investigation of a dendrogram is subjective and also tedious. That is why we wanted to see whether the CVIs and the gap statistic would be suitable to automate the cluster retrieval more objectively (research question RQ3). Figure 3 shows the relation between varying values of $k \in [2, N-1]$ (where N is the number of paths in our data set) and the different CVIs and the gap statistic, respectively. To get a consistent interpretation of the y axis (that is, the larger the value, the better), we inverted all CVIs that are to be minimized (namely, *COP*, *DB*, and *DB**) by subtracting their true values from one. In addition, we normalized their values by the z-score normalization. The z-score, z, of a raw score, x, is the number of standard deviations S by which x deviates from the mean value, $\bar{x}$, of the set of observed values: $z = \frac{x-\bar{x}}{S}$. It is a common practice in statistics to enable meaningful comparisons between two metrics with different value ranges.

If a CVI is just a strictly monontonic function of k, the best value for k is either the first or last value depending upon whether the function increases or decreases. If so, the CVI is not worth the effort to compute it. The curve of CVIs, however, has frequently been described in the literature to follow an 'elbow' shape, that is, a CVI decreases monotonically as k increases and from some k onwards the decrease of the curve flattens markedly (see Section III-C). If that is the case, one would pick the k where the curve starts to flatten (the elbow). There could also be other kinds of curves, for instance, a flipped U shape as, for instance, for normal distributions, in which case one would pick the k where the curve reaches its maximum. Ideally, different CVIs would suggest similar values for the optimal k so as to re-confirm each other. The reality of the CVIs we investigated for our paths, however, is not so simple as Figure 3 shows. Figure 3 omits the plots for hierarchical clustering with average linkage, denoted by h_a in the following, for reasons of space. The plots are similar to hierarchical clustering with complete linkage, denoted by h_c. We will note the differences explicitly in the remainder.

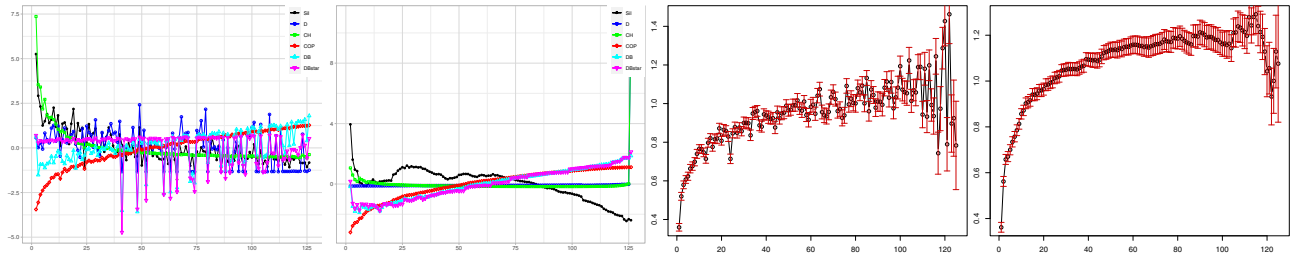
If we compare the CVIs for h_c and h_a to those of partitional clustering, referred to as p, we can immediately see that most CVIs show much more fluctuation for p than for h_a or h_c. Several CVIs may change drastically from one k to the immediate next one and then again to the next but one. Fluctuation of CVIs has been reported previously in other domains [36]. Most CVI measure somehow the cohesion within a cluster, on one hand, and the separation between clusters, on the other hand. Suppose we had three adjacent clusters A, B and C in one data set. If they were clustered into $\{A, B, C\}$ their separation would be minimized or if clustered into $\{A\}$, $\{B\}$, $\{C\}$ instead, their cohesion would be maximized. However, clustering them into $\{A \cup B, C\}$ or $\{A, B \cup C\}$ may make a great difference for the CVI and may cause the CVI to fluctuate even though all the three clusters may be very close to each other. That is, the influence of data dispersion between clusters may explain fluctuation. In our case, however, we do not only observe a fluctuation of a CVI for varying values of k, but also a drastic change of fluctuation of the same CVI between hierarchical and partitional clustering. We can only speculate why that is the case, but maybe that is due to the nature of partitional clustering which has an element of randomness included when the first k elements are selected for the initial clustering that is then being incrementally improved (see Section III-B). The recommendations that follow from this observation for future partitional clustering applications for timed paths is to run the partitional clustering for the same k several times with different random seeds and also to always evaluate immediately neighboring values for k so that there is no gap in the range that may otherwise suggest a wrong k.

Another observation that can be made immediately and contradicting statements in the existing literature on clustering is that the presumed 'elbow' shape suggesting a suitable value for k can generally not truly be observed in our results. We gathered the CVIs for the complete value range from two to the total number of paths minus one. The extreme values, however, are not really useful. What would be the point of clustering a data set with N elements into $2+i$ or $N-1-i$ clusters (with $0 \leq i \ll N$ being a value close to 1) clusters? So if we ignore the first and last few data points, we cannot really spot any clear indication of an 'elbow'. D and *CH* remain more or less the same and all other CVIs—except maybe *Sil*— are mostly montonically increasing (admittedly some of them with fluctuations even for h_a and h_c).

It is also interesting to note that *DB* and *DB** are almost alike for h_a and h_c, which suggests that they are interchangeable. For p, however, they behave differently: *DB** is a relative stable constant value for $k<40$ whereas *DB* increases with some fluctuation in that range. They both become more similar for $40 \leq k \leq 105$ also with regard to their fluctuation patterns. At about $k=105$, they start to diverge again. The mathematical difference between *DB* and *DB** was already discussed in Section III-B. In summary, they differ in the way they measure and weigh cohesion and coupling of clusters to each other. Whatever property of partitional clustering applied to our data set lets this mathematical difference become effective, in the middle range with less extreme values for k, none of them gives any clear hint on which value of k to select. Worse, for partitional clustering both CVIs fluctuate strongly in this middle range. For the hierarchical clustering, they neither appear to be useful as their curve increases monotonically.

The arguably more interesting curve, in particular for h_c, is the one of *Sil* because it is not monotonic. The graph in Figure 3b distinctively suggests $k=27$ as the best choice. For p and h_a, there is no such clear "winner"; $k=27$ could be a good choice for p and also h_a. For h_a, 49, 54, and 59 could be equally good alternatives, but there is very little difference in the range $25 \leq k \leq 60$ overall for h_a.

To calculate the *gap statistic*, we used function *clusGap*



(a) CVIs for p (b) CVIs for h_c (c) Gap statistics for p (d) Gap statistics for h_c

Fig. 3: Normalized CVIs and Gap statistics of partitional clustering p and complete-linkage hierarchical clustering h_c (the curves for h_a are very similar to those of h_c except for *Sil*, which has this interesting increase and decrease in the range $k\in[23,45]$, while h_a is largely flat there).

of the R package *cluster* [37]. *clusGap* calculates the gap statistics as described by Equation 3 in Section III-C. It determines the null reference distribution (see Section III-C) via bootstrapping— also known as *Monte Carlo simulation*. We used 60 as the number of Monte Carlo ("bootstrap") samples. Figure 3c–3d plots the obtained means (shown as dots) and standard deviations (shown as whiskers around the means) of the sampled comparison for varying values of k. Similarly as for the CVIs, the gap statistics tends to fluctuate more for partitional clustering, p, than for the hierarchical clusterings, h_a and h_c. Furthermore we can see that the gap and its standard deviation increase with increasing k, for all types of clusterings, until k get close to the total number of paths, N, where the gap decreases again, yet accompanied by a drastic growth of the standard deviation. As discussed above for CVIs already, values for k closer to 1 and N have little practical relevance, but here value ranges of k close to N are even more questionable because they show higher uncertainties. For all types of clustering, larger gaps can be obtained for $80\leq k\leq 100$.

The gap statistics has its own distinct way of measuring the quality of a cluster expressed as the pooled within-cluster distances around the cluster mean, W, described in Equation (2). The underlying idea of the gap statistic to compare the truly observed values to a null reference distribution could be applied to other cluster-quality measures, that is, the other CVIs described here, too. Unfortunately, *clusGap* does not offer that.

If we count the clusters in Figure 1 that fall into category (1), (2), or (4) and are maximal (that is, do not subsume any other counted cluster; marked by filled magenta rectangles), we find 26 non-overlapping clusters with at least two elements. All other 51 elements not included in those clusters should form a singleton cluster of their own. Thus, we expect $k=26+51=77$ many clusters. The gap statistic shown in Figure 3d supports this value. The values suggested by *Sil*, on the other hand, are far from 77. However, the external CVIs (described in Section III-C) when we compare the clusters obtained from the manual inspection of the dendrogram and the automated hierarchical clustering with complete linkage are as follows: *ARI* = 0.58, *JI* = 0.42, *FM* = 0.58, and *VI* = 0.20 indicating some level of similarity between the two but also a substantial difference. Only *RI* = 0.99 shows a high overlap, but as we discussed in Section III-C *RI* cannot fully be trusted if the data are imbalanced and k approaches N. Its high value is also relativized by the moderate value of *ARI*. These observations shed some doubt that the clusters can be retrieved by simply specifying a cut-off distance threshold implied by a constant k.

V. Conclusions

This section provides answers to our research questions.

(RQ1) Does clustering support us in investigating similar movement behaviour? Yes: it reduces the number of necessary comparisons drastically. If one has N paths, theoretically $N(N-1)/2$ comparisons would be necessary. Hierarchical clustering offers an automated way to group paths that are closer to each other. One can investigate closer paths in the dendrogram in a bottom up approach until one reaches questionable clusters. This process has worked very well in our study. Partitional clustering is less well suited because one must know the expected number of clusters in advance.

(RQ2) Is dynamic type warping (DTW) a suitable technique to determine the distances between paths? Mostly yes: DTW is able to align paths successfully by stretching or compressing time. Paths whose distances were reported low by DTW in our study were in fact found to be similar by us and worth to be inspected more closely. Yet it may still need some refinements to better capture human notion of similar paths, for instance, with respect to path subsumption and the weight of the different axes. Regarding computational scalability, we found that although the alignment, distance, and clustering of paths and the calculation of the CVIs involve heavy computation, overall the clustering of 127 paths over the complete range of possible values for k in our study was still tractable (besides some presumed implementation deficiency we encountered for hierarchical clustering).

(RQ3) Can classic clustering validation indices give us hints on "good" clusters? No: none of those gave us clear and correct hints on the number of clusters. There are many other indices proposed in the literature we have not explored and one of those might be better suited. Yet, our inspection of the dendrogram showed that it supports a manual validation quite well, so the need for completely automating this step may be less urgent.

References

[1] M. Steinbeck, R. Koschke, and M. Rüdel, "Movement patterns and trajectories in three-dimensional software visualization," in *International Working Conference on Source Code Analysis and Manipulation*, 2019, pp. 163–174.

[2] R. Koschke, *Software Engineering: International Summer Schools*. Springer, 2009, ch. Architecture Reconstruction, pp. 140–173, edited by Andrea De Lucia and Filomena Ferrucci.

[3] S. Bouktif, Y. Gueheneuc, and G. Antoniol, "Extracting change-patterns from CVS repositories," in *Working Conference on Reverse Engineering*, 2006, pp. 221–230.

[4] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: Experimental comparison of representations and distance measure," in *International Conference on Very Large Data Bases*, 2008, pp. 1542–1552.

[5] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, "The UCR time series classification archive," Oct. 2018, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[6] M. Abdelkader and M. Mimoun, "Clone detection using time series and dynamic time warping techniques," in *Third World Conference on Complex Systems*, 2015, pp. 1–6.

[7] C. Amornbunchornvej and T. Berger-Wolf, "Framework for inferring following strategies from time series of movement data," *ACM Transactions on Knowledge Discovery from Data*, vol. 14, no. 3, May 2020.

[8] V. Velichko and N. Zagoruyko, "Automatic recognition of 200 words," *International Journal of Man-Machine Studies*, vol. 2, pp. 223–234, 1970.

[9] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *International Congress on Acoustics*, vol. 3, 1971, pp. 65–69.

[10] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.

[11] C. Myers, L. Rabiner, and A. Rosenberg, "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 6, pp. 623–635, 1980.

[12] T. Giorgino, "Computing and visualizing dynamic time warping alignments in R: The dtw package," *Journal of Statistical Software*, vol. 31, no. 7, pp. 1–24, 2009. [Online]. Available: http://www.jstatsoft.org/v31/i07/

[13] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

[14] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[15] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster analysis*. Wiley, 2011.

[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009, ch. 14.3, Cluster Analysis.

[17] O. Arbelaitz, I. Gurrutxaga, Muguerza, J., J. M. Perez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, vol. 46, no. 1, pp. 243–256, 2013.

[18] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[19] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, vol. 78, no. 383, pp. 553–569, 1983.

[20] M. Meila, "Comparing clusterings by the variation of information," in *16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel*. Springer, 2003, pp. 173–187.

[21] C. Sugar, "Techniques for clustering and classification with applications to medical problems," PhD Dissertation, Stanford University, 1998.

[22] C. Sugar, L. Lenert, and R. Olshen, "An application of cluster analysis to health services research: empirically defined health states for depression from the sf-12," Stanford University, Technical Report, 1999.

[23] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of data clusters via the gap statistic," *Journal of the Royal Statistical Society B*, vol. 63, no. Part 2, pp. 411–423, 2001.

[24] W. Gaul and D. Pfeifer, Eds., *From Data to Knowledge*. Springer, 1996, ch. Null models in cluster validation (by A. Gordon).

[25] A. Sardá-Espinosa, "Time-series clustering in R using the dtwclust package," *The R Journal*, 2019.

[26] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[27] J. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters,," *Journal of Cybernetics*, vol. 3, pp. 32–57, 1973.

[28] I. Gurrutxaga, I. Albisua, O. Arbelaitz, J. Martín, J. Muguerza, and I. Pérez, J.M. Perona, "SEP/COP: an efficient method to find the best partition in hierarchical clustering based on a new cluster validity index," *Pattern Recognition*, no. 43, pp. 3364–3373, 2010.

[29] D. Davies and D. Bouldin, "A clustering separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 224–227, 1979.

[30] M. Kim and R. S. Ramakrishna, "New indices for cluster validity assessment," *Pattern Recognition Letters*, vol. 26, no. 15, pp. 2353–2363, 2005.

[31] T. Calinski and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics*, vol. 3, pp. 1–27, 1974.

[32] R. Koschke and M. Steinbeck, "Replication package," **uploaded to EasyChair for our reviewers**; later published under link.will.be. available.when.accepted, Jun. 2020.

[33] D. H. R. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, Sep. 2006.

[34] "Axivion gmbh." [Online]. Available: http://www.axivion.com/en/

[35] "Embedded software engineering," Dec. 2019. [Online]. Available: https://ese-kongress.de

[36] X. Wang and Y. Xu, "An improved index for clustering validation based on silhouette index and calinski-harabasz index," *IOP Conference Series: Materials Science and Engineering*, vol. 569, p. 052024, Aug. 2019.

[37] M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik, *cluster: Cluster Analysis Basics and Extensions*, 2019, R package version 2.1.0.