SUPPLEMENTARY MATERIAL
Propagating Visual Designs to Numerous Plots and Dashboards

Saiful Khan, University of Oxford, UK
Phong H. Nguyen, Redsift Ltd., UK
Alfie Abdul-Rahman, King's College London, UK
Benjamin Bach, University of Edinburgh, UK
Min Chen, University of Oxford, UK
Euan Freeman, University of Glasgow, UK
Cagatay Turkay, University of Warwick, UK

A ALGORITHMS FOR COMPUTING SIMILARITY MATRICES

In this section we will describe the process of computing similarity matrices $\boldsymbol{S}_{rd}$ and $\boldsymbol{S}_{dd}$ defined in the paper and shown in Fig. 8. Matrix $\boldsymbol{S}_{rd}$ represents the pairwise similarity between the reference data streams and discovered data streams, and matrix $\boldsymbol{S}_{dd}$ represents pairwise similarities between each discovered data stream.

As described in the paper, each data stream (an OntoData instance) has several attributes: i.e., description, keywords, data type and API endpoint. Each attribute is a feature and we denote the number of features as f. The reference data streams (an ordered list) is denoted as a matrix $\boldsymbol{R} \in \mathbb{R}^{k \times f}$, where k is the number of reference data streams. Discovered data streams are denoted as a matrix $\boldsymbol{D} \in \mathbb{R}^{n \times f}$, where n is the number of discovered data streams.

A.1 Computing Matrix S_{rd}

The ordering algorithm will use $\boldsymbol{S}_{rd}$ to sort the data streams within each group. To compute $\boldsymbol{S}_{rd}$, we first create a similarity matrix for each data stream feature and discovered data streams, then aggregate the four matrices.

Data type similarity. We derive a feature vector $\boldsymbol{r}^{(t)} \in \mathbb{R}^k$, where $\boldsymbol{r}^{(t)}$ corresponds to the data type column of $\boldsymbol{R}$; $\boldsymbol{r}^{(t)} = [r_1^{(t)}, r_2^{(t)}, \dots r_k^{(t)}]$. We derive another feature vector $\boldsymbol{d}^{(t)} \in \mathbb{R}^n$, where $\boldsymbol{d}^{(t)}$ corresponds to the data type column of $\boldsymbol{D}$; $\boldsymbol{d}^{(t)} = [d_1^{(t)}, d_2^{(t)}, \dots d_n^{(t)}]$.

A function ϕ computes the pairwise similarity matrix between the feature vectors $\boldsymbol{r}^{(t)}$ and $\boldsymbol{d}^{(t)}$ and is defined in Equation 1.

$$\phi(\boldsymbol{r}^{(t)}, \boldsymbol{d}^{(t)}) \in \mathbb{R}^{k \times n} = \begin{cases} 1, & \text{if data types are similar} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Keyword similarity. Similar to the data type vectors, we derive keyword feature vectors $\boldsymbol{r}^{(w)}$ and $\boldsymbol{d}^{(w)}$ which corresponds to the keyword column of matrix $\boldsymbol{R}$ and $\boldsymbol{D}$ respectively. The keywords attribute of a data stream contains a subset of a set of all keywords used to define data streams in the system. Therefore, Jaccard [17, 35] similarity measurement function is used to compute the pairwise similarity matrix. A function ψ computes the size of the intersection divided by the size of the union of two keywords sets, defined in Equation 2.

$$\psi(\boldsymbol{r}^{(w)}, \boldsymbol{d}^{(w)}) \in \mathbb{R}^{k \times n} = \frac{(\boldsymbol{r}^{(w)} \cup \boldsymbol{d}^{(w)})}{(\boldsymbol{r}^{(w)} \cap \boldsymbol{d}^{(w)})} \tag{2}$$

Description similarity. A description field is free-form text and can be represented as a collection of words or terms. Therefore, term frequency (tf) and inverse document frequency (idf) similarity measurement algorithms [27] will be suitable here.

We derive feature vectors $\boldsymbol{r}^{(d)}$ and $\boldsymbol{d}^{(d)}$ which corresponds to the description column of $\boldsymbol{R}$ and $\boldsymbol{D}$ respectively.

Next, we derive a matrix $\boldsymbol{U}$, where each vector $\boldsymbol{u}_i \in \boldsymbol{U}$ is a tf-idf vector [27] of i-th element of $\boldsymbol{r}^{(d)}$. Similarly, we derive another matrix $\boldsymbol{V}$, where each vector $\boldsymbol{v}_j \in \boldsymbol{V}$ is a tf-idf vector of j-th element $\boldsymbol{d}^{(d)}$.

A function ω computes similarity by measuring Cosine similarity [27] between $\boldsymbol{U}$ and $\boldsymbol{V}$; defined in Equation 3.

$$\omega(\boldsymbol{r}^{(d)}, \boldsymbol{d}^{(d)}) \in \mathbb{R}^{k \times n} = \frac{\boldsymbol{U}\boldsymbol{V}}{\|\boldsymbol{U}\|\|\boldsymbol{V}\|} \tag{3}$$

Given any two row vectors $\boldsymbol{u} \in \boldsymbol{U}$ and $\boldsymbol{v} \in \boldsymbol{V}$ the Cosine similarity between the vectors can be computed by Equation 4.

$$\frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|} = \frac{\sum_{x=1}^{q} u_x v_x}{\sqrt{\sum_{x=1}^{q} u_x^2}\sqrt{\sum_{x=1}^{q} v_x^2}} \tag{4}$$

where u_x and v_x are components of vector $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively; and q is the number of components (all possible words from the description fields).

API endpoint similarity. We derive two feature vectors $\boldsymbol{r}^{(a)}$ and $\boldsymbol{d}^{(a)}$, which correspond to the API endpoint attribute columns of $\boldsymbol{R}$ and $\boldsymbol{D}$ respectively. RESTful API endpoints contain textual features such as a data stream server address, API route, and URL encoded parameters. An API endpoint can provide information about a data stream, such as its data product, component, source, and type (described in Section 4). We tokenize the endpoint fields to extract their terms (words) and use similar functions used for description field similarity measurement, $\omega(\boldsymbol{r}^{(a)}, \boldsymbol{d}^{(a)}) \in \mathbb{R}^{k \times n}$, to compute a similarity matrix.

Aggregated similarity. We aggregate the four similarity matrices computed above. An aggregation function γ, defined in Equation 5, computes the aggregated matrix, $\boldsymbol{S}_{rd} \in \mathbb{R}^{k \times n}$. This function takes a weighting average of the input matrices.

$$\begin{aligned} \boldsymbol{S}_{rd} = \gamma(\boldsymbol{R}, \boldsymbol{D}) = [\alpha\psi(\boldsymbol{r}^{(w)}, \boldsymbol{d}^{(w)}) + \beta\omega(\boldsymbol{r}^{(d)}, \boldsymbol{d}^{(d)}) \\ + \theta\omega(\boldsymbol{r}^{(a)}, \boldsymbol{d}^{(a)})] \odot \phi(\boldsymbol{r}^{(t)}, \boldsymbol{d}^{(t)}) \end{aligned} \tag{5}$$

where α, β, and θ are scalar constants that define the relative weights of keywords, description, and endpoint fields in the similarity measurement, where $\alpha + \beta + \theta = 1$. The pairwise similarity between any data type field is 0 or 1; therefore, for the keywords type we use the element-wise product (or Hadamard product), $\odot$, in the aggregation function.

A.2 Computing Matrix S_{dd}

The matrix $\boldsymbol{S}_{dd} \in \mathbb{R}^{n \times n}$ computes pairwise similarities between each discovered data streams. We use this matrix for creating uniform groups of similar data streams.

Computation of the matrix $\boldsymbol{S}_{dd}$ is almost similar to the computation steps applied for deriving the matrix $\boldsymbol{S}_{rd}$ in previous Section A.1. For each feature, e.g., keyword, description, and API endpoint of the matrix $\boldsymbol{D}$, we derive three similarity matrices: $\psi(\boldsymbol{d}^{(w)}, \boldsymbol{d}^{(w)})$, $\omega(\boldsymbol{d}^{(d)}, \boldsymbol{d}^{(d)})$, and $\omega(\boldsymbol{d}^{(a)}, \boldsymbol{d}^{(a)})$ (using Equation 2 and 3). Finally, the aggregation function, $\lambda(\boldsymbol{D}, \boldsymbol{D})$ aggregates the three matrices, defined in Equation 6. A group can consist of data stream of different data types; therefore, we excluded the data type feature from the computation of $\boldsymbol{S}_{dd}$.

$$\begin{aligned} \boldsymbol{S}_{dd} = \lambda(\boldsymbol{D}, \boldsymbol{D}) = \alpha\psi(\boldsymbol{d}^{(w)}, \boldsymbol{d}^{(w)}) + \beta\omega(\boldsymbol{d}^{(d)}, \boldsymbol{d}^{(d)}) \\ + \theta\omega(\boldsymbol{d}^{(a)}, \boldsymbol{d}^{(a)}) \end{aligned} \tag{6}$$

B ALGORITHMS FOR GROUPING AND ORDERING

Algorithm 1 outlines the brute-force grouping approach described in Section 7.2. Algorithm 2 outlines the graph spectral grouping approach described in Section 7.2. Algorithm 3 outlines the procedure for sorting and ranking groups.

Algorithm 1: A brute-force algorithm for grouping data streams

Input: A matrix $\boldsymbol{S}_{dd} \in \mathbb{R}^{n \times n}$; size of each group k
Output: Groups $\boldsymbol{G} \in \mathbb{R}^{k \times m}$

$n \leftarrow \|\boldsymbol{S}_{dd}\|$
/* Number m groups to construct */
$m = \lfloor \frac{n}{k} \rfloor$
$\boldsymbol{G} \leftarrow Null$
$p \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
  $visited[i] \leftarrow False$
end
for $i \leftarrow 1$ **to** n **do**
  if $visited[i] == True$ **then**
    continue
  else
    /* Indices of k max items of i-th row of $\boldsymbol{S}_{dd}$ */
    $max\_k\_idx \leftarrow \texttt{MaxIndices}(\boldsymbol{S}_{dd}[i], k)$
    $\boldsymbol{G}[p] \leftarrow max\_k\_idx$
    $p \leftarrow p + 1$
    /* Mark the grouped items as visited */
    for $j \leftarrow 1$ **to** k **do**
      $idx \leftarrow max\_k\_idx[j]$
      $visited[idx] \leftarrow true$
    end
  end
end

Algorithm 2: Spectral graph partitioning algorithm for grouping data streams

Input: A similarity matrix $\boldsymbol{S}_{dd} \in \mathbb{R}^{n \times n}$; size of each group k
Output: Groups $\boldsymbol{G} \in \mathbb{R}^{k \times m}$

- Number m groups to construct, $m = \lfloor \frac{n}{k} \rfloor$.
- Compute diagonal matrix $\boldsymbol{D}$ of $\boldsymbol{S}_{dd}$.
- Compute Laplasian matrix, $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{S}_{dd}$.
- Given m number of groups to create; compute the first m eigenvectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2 \ldots \boldsymbol{v}_m$ of $\boldsymbol{L}$, such that $\boldsymbol{L}_v = \lambda \boldsymbol{D}_v$.
- Let $\boldsymbol{V} \in \mathbb{R}^{n \times m}$ be the matrix containing the vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\ldots \boldsymbol{v}_m$ as columns.
- For $i = 1, 2, \ldots n$, let $\boldsymbol{d_i} \in \mathbb{R}^m$ be the vector corresponding to the i-th row of $\boldsymbol{V}$.
- Finally, use K-means algorithm to cluster $\boldsymbol{d_1}$, $\boldsymbol{d_2}$, $\ldots$, $\boldsymbol{d_n}$ into m groups $\boldsymbol{g_1}, \boldsymbol{g_2}, \ldots \boldsymbol{g_m} \in \boldsymbol{G}$ and $\boldsymbol{G} \in \mathbb{R}^{k \times m}$.

Algorithm 3: Algorithm for sorting and ranking groups

Input: Groups $\boldsymbol{G} \in \mathbb{R}^{k \times m}$; matrix $\boldsymbol{S}_{rd} \in \mathbb{R}^{k \times n}$
Output: Sorted and ranked $\boldsymbol{G}' \in \mathbb{R}^{k \times m}$

$m \leftarrow \|\boldsymbol{G}\|$
$k \leftarrow \|\boldsymbol{S}_{rd}\|$
/* Priority queue for queuing based on ranking */
$\boldsymbol{G}' \leftarrow Null$
/* For each group */
for $i \leftarrow 1$ **to** m **do**
  $group \leftarrow \boldsymbol{G}[i]$
  $group\_sorted \leftarrow Null$
  $group\_score \leftarrow 0$
  for $j \leftarrow 1$ **to** k **do**
    $col\_idx \leftarrow group[j]$
    $col\_vec \leftarrow \boldsymbol{S}_{rd}[:, col\_idx]$
    /* The index of max element in col_vec */
    $row\_idx \leftarrow \texttt{MaxIndices}(col\_vec, 1)$
    $similarity \leftarrow col\_vec[row\_idx]$
    $group\_score \leftarrow group\_score + similarity$
    $group\_sorted[row\_idx] \leftarrow col\_idx$
  end
  /* Add the sorted group to priority queue */
  $\texttt{PriorityQAdd}(\boldsymbol{G}', group\_sorted, group\_score)$
end

C Propagating Dashboards with Links

In the paper, we described the process for propagating a visualization function with multiple data streams. Propagating a dashboard is more complicated, because of the need to match data streams (OntoData instances) *and* web page links (OntoPage instances). If incorrectly matched, the visual designs in the dashboard would not be linked to the correct webpage.

Fig. 9 shows the process used to propagate a dashboard with data streams and links. After the infrastructure manager selects a VIS function for propagation, its reference data streams and links are retrieved from the ontology. The metadata of the data streams and links are forwarded to the UI. While, the process of propagation for data streams is described in Section 7, the process of propagating links involves additional steps.

Links are web pages (i.e., OntoPage instances) and their attributes include a VIS function and data streams (described in Section 5.1). Following the process described earlier in Section 6.1, the infrastructure manager formulates a search query and discover a possible list of data streams. From the discovered data streams, a function (a) creates possible groups of data streams and orders each group (as described in Section 7 and Section A). For discovered data streams, another function scans the ontology to retrieve (b) all possible pages or bindings visualizing the groups of (a). From the (a) and (b) list of possible groups, both data stream groups and page groups are created. We then use similar grouping, ordering, and ranking algorithms (described in Section 7 and Section A).

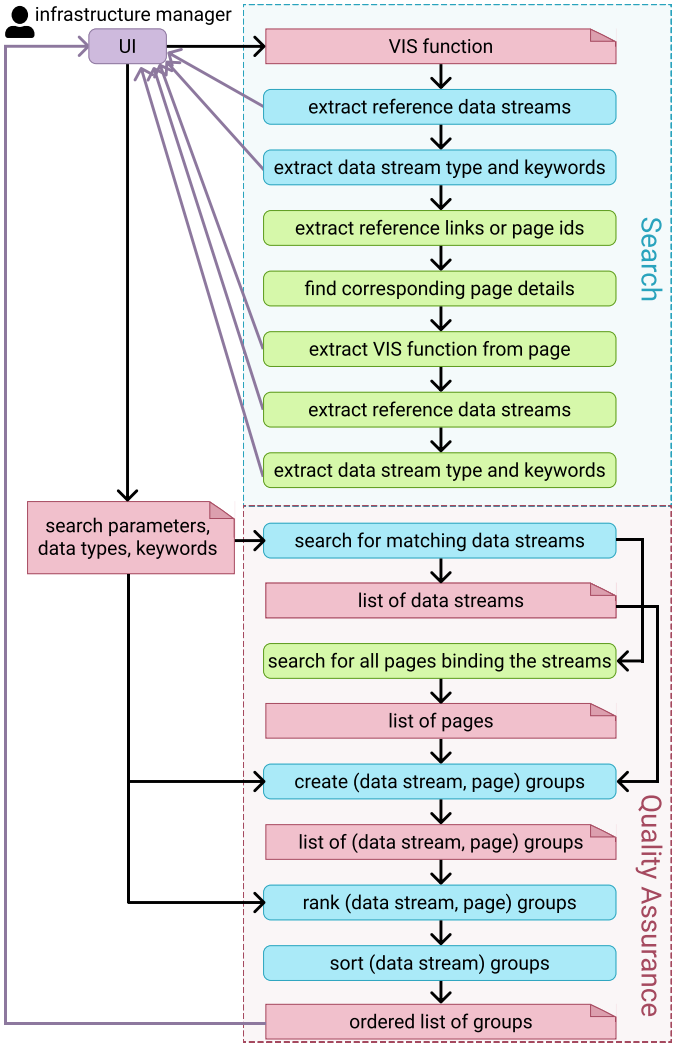

Fig. 9. A flowchart illustrating a search and ranking workflow for finding relevant links and propagating the links to a dashboard.