Schedae Informaticae Vol. 32 (2023): 9–25
doi: 10.4467/20838476SI.23.001.19323

Measuring Similarity Between ETL Processes Using Graph Edit Distance[1]

Maciej Brzeski[1,2], Adam Roman[1]
[1]Faculty of Mathematics and Computer Science, Jagiellonian University,
Łojasiewicza 6, 30-348 Kraków, Poland,
[2]Informatica Polska, Kamienna 21, Kraków, Poland,
e-mail: *maciej.brzeski@doctoral.uj.edu.pl, adam.roman@uj.edu.pl*

Abstract. Maintaining data warehouses and ETL processes is becoming increasingly difficult. For this reason, we introduce a similarity measure on ETL processes, based on the edit distance of a graph, which models the process. We show both the exact way how to calculate it and heuristic approaches to compute the estimated similarity more quickly. We propose methods to improve graph edit distance based on the assumption that the ETL process model is a directed acyclic graph.

Keywords: ETL process, ETL similarity, graph edit distance, error-tolerant graph matching

1. Introduction

A **data warehouse** is a large, centralized repository of data that is specifically designed to support business intelligence (BI) activities such as data analysis, reporting, and data mining. It is a complex, integrated system that collects, stores, and manages data from various sources, including operational systems, external sources, and other data warehouses.

The data in a warehouse is organized and structured in a way that makes it easy for users to access and analyse, regardless of its original format or location. This is achieved through a process known as ETL (Extract, Transform, Load), which

[1]This work is the result of research project no. DWD/4/66/2020 supported and funded by Ministry of Education and Science in Poland.

involves extracting data from multiple sources, transforming it into a standardized format, and loading it into the warehouse.

Data warehouses are used by organizations of all sizes and across a variety of industries to support decision-making processes and gain valuable insights into business operations. They enable organizations to store, analyse, and act on vast amounts of data in real-time, helping to drive business growth and success. This is a critical component of modern data management.

One of the most growing areas of data management nowadays is so-called **data governance**. The key element of data governance is the automatic analysis of data flow, or so-called Data Lineage. There are several reasons for this. In industries such as banking and financial services, a major factor is the regulations that have been growing for many years (BCBS 239 standard [33], DFAST – Dodd-Frank Act Stress Tests, CCAR – Comprehensive Capital Analysis and Review). Also outside the financial sector, Data Lineage, i.e. knowledge of how data is processed in an organization, always supports the efficiency of problem analysis work and planning for future changes (so-called impact analysis).

In the area of Data Lineage, one of the most important trends that has been observed for several years now is the increasing amount of information available. Thanks to advanced parsing and analysis techniques, organizations are able to gather huge amounts of information on how data are processed and stored. However, it is becoming increasingly challenging to analyse this information, and thus finding among the millions of connections those that are relevant for solving a particular problem.

One of the biggest challenges in this area is data flow analysis in multi-level graphs modelling ETL processes, spatial distribution of flows containing tens of thousands of processes, and automatic similarity analysis of hundreds of thousands of complex data processing workflows. These are challenges where, like nowhere else, cooperation between science and business is crucial, because such industrial-scale data flow analysis involves analysing problems for which there are currently no ready-made solutions [1].

Although ETL processes can be defined using the GUI, ultimately the process must be converted to executable code. This code must be characterized not only by an appropriate level of correctness (functional suitability, functional correctness, functional appropriateness), but also by an appropriate level of non-functional characteristics, in particular maintainability, or more precisely – its sub-characteristics: modifiability, analysability, and reusability (cf. ISO 25010 quality model [24]).

Maintainability of the ETL workflows is crucial for the cost of the ETL lifecycle. Maintainability can be improved by using static analysis [23] and depends on a number of parameters, e.g., the cost of handling evolution events during the ETL lifecycle [41]. In particular, the code implementing the ETL process should be free of code smells that increase technical debt, which negatively affects the cost of maintenance. One of the well-known code smells is code duplication, which violates the DRY rule (Don't Repeat Yourself). This code smell can be analysed in relation to not a single ETL process, but to a collection of such processes.

One of the key metrics to help identify potential repetitions or, in general, similarities between individual fragments of two ETL processes is the process similarity measure. Searching for similarities in ETL processes has the following advantages:

- it allows us to increase maintainability by including repetitive code in a separate function and referring in other (parts of) ETL processes to this subprocess; if some change is needed in this subprocess, it will have to be done only in one place, and not in many;
- it allows us to determine whether a given two ETL processes are similar or even identical;
- it facilitates process definition by, for example, suggesting to the developer certain functions to use, e.g., if the system recognizes that the process under construction is identical to a part of an existing process in the database, it can prompt the user whether they want to implement this very process, which will save time and avoid implementation errors;
- it makes it easier to analyse and read ETL processes by simplifying their structure that references to subprocesses.

In this paper, we show how to calculate the similarity between the ETL processes, and we propose some heuristic approaches to count the estimated similarity more quickly. The structure of this paper is as follows. In section 2. we formalize the ETL processing in more detail, presenting the tool called MetaDex, used to extract metadata from various sources to obtain the lineage. In section 3. we introduce the concept of a graph edit distance, which will be used in our approach. Section 4. discusses the related work on the ETL similarity and the Graph Edit Distance application. In section 5. we define the ETL process using the graph theory. We also show how to adapt MetaDex to our model. Section 6. shows how to use Graph Edit Distance to solve the ETL similarity problem. In section 7. we present the experiments and their results on the real ETL processes. Section 8. follows with the conclusions.

2. ETL Process Modelling

The proper modelling of the ETL process is essentially the foundation for the success of many data warehousing projects, as the ETL processes form the core of data warehousing architectures. There is no standard model for the representation of this process. Several researchers proposed some modelling techniques based on various formalisms, such as unified modelling language (UML) [43], conceptual constructs [39, 42], ontology [4], QoX-driven ETL modelling [44], OHM model [13], and graphical flow, which includes business process model notation (BPMN) [32, 37]. In [40] a graph model was proposed, similar to the one shown in the subsection 2.2.. Also, worth mentioning is the research in topic lineage tracing [12], where the authors are focusing on data that undergoes a sequence of transformations. A systematic literature review, with a more detailed analysis of the approaches and differences, is presented by Dhaouadi in [14].

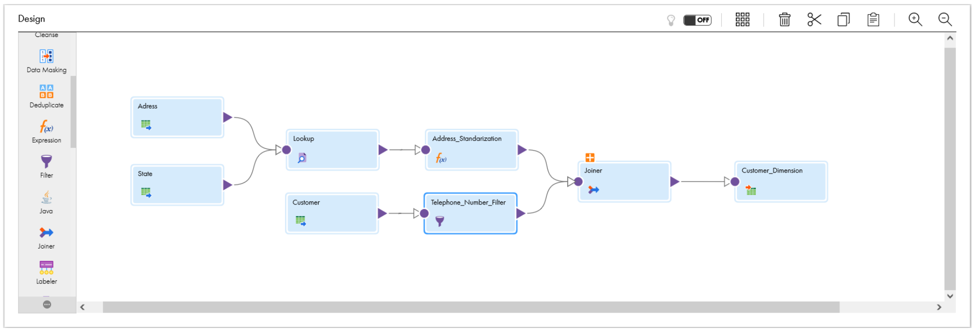

Figure 1.: Sample ETL job

2.1. ETL Process Granularity

The ETL process can be understood at different levels of generality. A general view speaks of data extraction, processing and recording. This consists of various procedures that call themselves at specific times or are called by other procedures. However, to support code maintenance tools, we need to look at what the data look like at the most granular level, where data operations are directly defined. Figure 1 shows an example of the procedure, written using the Informatica Intelligent Cloud Services [22].

There are many ETL tools, as well as mainframe systems, data manipulation languages (e.g. SQL), and general-purpose languages that can be used for data processing. However, different ETL tools allow different operations and work with different models of the ETL processes. Therefore, a common model, which standardizes this, is needed.

2.2. MetaDex

MetaDex is a tool used to extract metadata from various sources to obtain lineage. It automatically analyses data processing, extracts the most granular metadata and tracks data dependencies across data sources. MetaDex is currently a product of Informatica, but its concept was demonstrated by Duda et al. in 2012 [18]. MetaDex consists of several components, but for us, the most important is the common representation of ETL processes. A simple example of the MetaDex use for the sample SQL procedure, shown in Listing 1, is presented in Figure 2.

Listing 1: Example of a SQL procedure

SELECT $2 \cdot a_0 + b_0$ INTO OUT (out_0) FROM A
JOIN B ON $a_1 = b_1$
WHERE $a_0^2 + b_0^2 = 1$

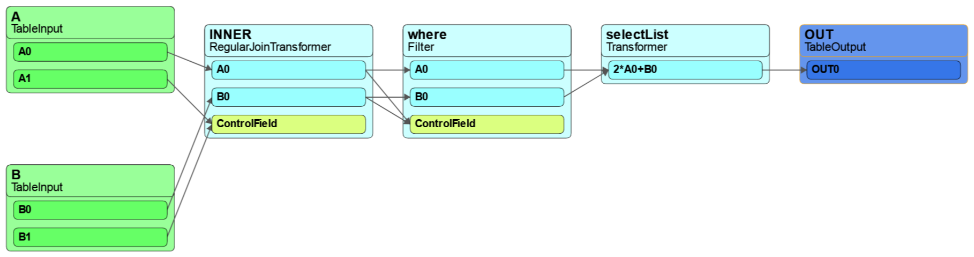

Figure 2.: Metadex model of the SQL procedure shown in 1

Detailed expressions, used in `join` and `where` conditions, are contained in the corresponding transformation, similarly to the expression $2 \cdot a_0 + b_0$ shown in the last transformation before output table.

The MetaDex model is represented by a nested graph, consisting of two levels. In the outer layer, we have a representation of the overall ETL workflows of the data stream, e.g., tables, transformations, filters or joins. Each vertex represents a state during the processing of a data stream. The inner layer provides a more detailed description of how the values for each field (column) are calculated. Each field contains an expression that depends on the fields found in the previous transformation. More formally, each expression is a calculation tree, where the root is the given field and the leaves are the fields from the source processing vertex. In addition, each transformation can have dedicated information, such as the type of join. Frequent information of the processing vertex is some expression, called control expression, which is not used for calculated value, but it impacts the whole data stream. Examples include the previously mentioned `join` and `filter` transformations, but can also include columns used in aggregation (e.g. `group by` in SQL) or other relevant expressions that affect the resulting data stream (but not its value). Therefore, two types of directed edges are distinguished:

- value, if the processing source has an impact on the values of the data stream, e.g., addition;

- control, if the processing source has an impact on the data stream, but not on values, e.g., data filtering.

The model tries to reflect as closely as possible the flow defined in the most popular ETL tools. Each transformation or processing of the data stream is usually represented by one outer vertex. The whole procedure is a directed, acyclic graph, where leaves must be sources (input tables, variables or constants), and roots must be targets (output tables). The model includes dozens of different data processing elements available in various ETL tools, as well as a diverse representation of input and output data. In the end, we have a universal representation of the ETL process, so we can much more easily compare processes with each other.

Table 1.: Table of permitted editing operations together with their costs

edit operation	edit cost
substitute α-labelled node by α'-labelled node	$c_V(\alpha, \alpha')$
delete isolated α-labelled node	$c_V(\alpha, \epsilon)$
insert isolated α-labelled node	$c_V(\epsilon, \alpha)$
substitute β-labelled edge by β' - labelled edge	$c_E(\beta, \beta')$
delete β-labelled edge	$c_E(\beta, \epsilon)$
insert β-labelled edge	$c_E(\epsilon, \beta)$

3. Graph Edit Distance

Graph Edit Distance is a generalization of Levenshtein distance [28]. In the case of graphs, both edges and isolated vertices are allowed to be inserted and removed. However, a problem defined in this way is NP-hard, even for uniform edit costs [45], and APX-hard for metric edit costs [29]. We consider directed, labelled (both vertex and edge) graphs. The following operations on the graph are defined:

Each such operation transforms a graph into another graph. We can consider a sequence of such operations. Our goal is to find an edit path, i.e. an operation path that transforms graph $\mathcal{G}_1$ into graph $\mathcal{G}_2$. Then the edit distance for the graphs $\mathcal{G}_1 = (V_1, E_1)$, $\mathcal{G}_2 = (V_2, E_2)$ is defined as

$$GED(\mathcal{G}_1, \mathcal{G}_2) = \min_{(e_1, \ldots, e_n) \in \gamma(\mathcal{G}_1, \mathcal{G}_2)} \sum_{i=1}^{n} c(e_i),$$

where $\gamma(\mathcal{G}_1, \mathcal{G}_2)$ is the set of all possible edit paths, transforming graph $\mathcal{G}_1$ into graph $\mathcal{G}_2$, and $c(e_i)$ is a cost of the i-th vertex (c_V) or edge (c_E) operation.

Despite the clear definition, it is unclear how to find a solution. The number of possible paths can be infinite, and even for a particular path one has to check whether two graphs are isomorphic [34], which is a difficult problem and to date, no polynomial solution is known. Instead, we can define the node map as follows.

Definition 1 (Node Map)**.** *Relation $\Pi \subset (V_1 \cup \{\epsilon\}) \times (V_2 \cup \{\epsilon\})$ is called node map, if it satisfies the following conditions:*

- $|\{v \mid v \in (V_1 \cup \{\epsilon\}) \wedge (u, v) \in \Pi\}| = 1$, *for every* $u \in V_1$
- $|\{u \mid u \in (V_2 \cup \{\epsilon\}) \wedge (u, v) \in \Pi\}| = 1$, *for every* $v \in V_2$

Intuitively, a node map is a bijection between graphs, except that we allow a vertex to be marked as inserted or deleted and assign ϵ to it. With node map, we can create an induced edit path. Directly from the definition, we get the needed operations on vertices. If both vertices of an edge e_1 have been mapped to the end of an edge e_2 in the second graph, then in the edit path we add substitution e_1 into e_2. Otherwise, we either insert an edge (when e_1 does not exist) or delete an edge (when e_2 does not exist). We must additionally request that the deletion of edges be done at the beginning and their addition at the end, so that non-isolated vertices are never deleted. Then, the edit distance can be defined as

$$GED(\mathcal{G}_1, \mathcal{G}_2) = \min\{c(P_\pi) \mid \pi \in \Pi(\mathcal{G}_1, \mathcal{G}_2)\},$$

where P_π is the edit path induced from the mapping π.

Various approaches have been tried, both to calculate the exact edit distance and the approximation distance. In Justice's paper [26] it was shown how the GED problem can be reduced to linear programming, for solving which many optimization tools exist. It is also popular to try to use tools for assignment problems. Riesen [35] suggested using the Hungarian algorithm to obtain an approximate solution. On the other hand, Bougleux [8] showed how to use quadratic assignment problem for solving graph edit distance. In addition, many heuristics have been developed and compared in the Blumenthal's paper [6]. Blumenthal, in his PhD dissertation [5], described and compared different approaches to calculating edit distance.

3.1. Linear Sum Assignment Problem

The Linear Sum Assignment Problem (LSAP), also known as the Assignment Problem, is a classic optimization problem in mathematics and operations research. It involves finding the optimal assignment of a set of jobs or tasks to a set of workers, machines, or resources, given a set of costs or penalties associated with each possible assignment. The problem has a polynomial solution, which can be found by the so-called Hungarian algorithm [27, 31]. Several improvements for this algorithm exist, such as the Jonker–Volgenant algorithm [25], or the use of algorithms to find the maximal flow like Ford–Fulkerson algorithm [20]. LSAP is a popular heuristic that can be used both to find an approximate solution [35] and a heuristic for finding an exact solution, using, for example, the A* algorithm [36]. This is because when we have two graphs with the identical number of vertices, and we ignore the cost of edge deletion, the GED is reduced to an assignment problem.

3.2. Linear Sum Assignment Problem with Error-Correction (LSAPE).

When the graphs are of different sizes, it is required to extend the LSAP problem with editions [7]. In the simplest solution [10], artificial vertices can be added to the second graph, which is responsible for deleting vertices from the first one (and vice-versa). Then, the assignment to the corresponding vertex is determined by the cost of deletion while preventing assignment to the remaining artificial vertices (e.g., by assigning the infinite weight). However, the cost matrix is largely filled with zeros (cost between artificial vertices in both graphs) and several modifications of the Hungarian algorithm have been developed for such a case [38, 9].

4. Related Work

4.1. ETL similarity

The topic of the similarity of ETL processes seems to be an important issue. Despite this, it has not been addressed very often. In [1], a high-level ETL process management system was proposed, which allows the user to find the most similar ETL processes. However, similarity itself was not defined, due to its difficulty. The paper [2] introduces a so-called *match* operator between two ETL processes. Predefined weights of interdependent transformation pairs were used in this work, and the similarity was calculated as the cosine of two vectors. To our best knowledge, it is the first approach using graph edit distance to measure the similarity of ETL processes.

4.2. Business Process Model similarity

The similarity of Business Process Models is a widely recognized and significant matter. It serves various purposes, including assessing the alignment between reference and actual models, identifying related models within a repository, and finding services that comply with a specification provided by a process model. A summary of the methods used in this problem is presented in the paper [17]. In [16] various techniques for defining BPM similarity were presented, such as node matching similarity, structural similarity and behavioural similarity. A comparison of different approaches is shown in the survey paper [3]. In the case of comparing ETL processes, it is much more important to find matched transformations than to calculate exact similarity. In addition, it is much more useful to have an accurate measure for similar processes, since they are mainly the ones that can be refactored.

Graph edit distance is mainly used where it is important for us to calculate fine-grained similarity with many, but rather small graphs. It has been successfully used for another similarity problem – malware detection in source code [19, 11]. The use of graph edit distance in comparing BPM was presented in several papers. The paper by Dijkman at al. [15] investigates the problem of ranking all process models according to their similarity to a given process model. Several approaches have been compared: greedy, exhaustive, heuristic and A*-based. The algorithm based on A* achieved the best results, being slower only than the greedy algorithm. Grigori at al. in their paper [21] develop matching techniques that operate on behaviour models for service discovery. They faced similar problems as in this work, such as model granularity or the non-trivial comparison of single vertices. The experiments focused on different cut-offs in a state-space searching algorithm similar to A*.

In both papers, authors focused more on the application and demonstration of the effectiveness of the A* algorithm. In contrast to their work, our goal is different. The calculation of graph edit distance can be very time-consuming, even for relatively small graphs with 20 vertices. We propose modifications to the GED algorithm that result in a significantly shorter runtime, especially when the ETL processes being compared are similar.

5. Model

5.1. Graph model

Creating the common representation of ETL processes is not a trivial task. It is also quite time-consuming to calculate the edit distance, so we will try to simplify the model to apply the GED directly. Moreover, by doing so, we will get a representation that will be much more flexible and adaptable to different models of the ETL-processes.

The model used to compare the graphs will be a directed graph $\mathcal{G} = (V, E)$. Instead of having vertex labels, we will expect to have for each vertex the cost of deleting it, and for each pair of vertices from two graphs we define a function that returns the cost of substitution one with the other. We allow two types of edge labels (corresponding to a value and control dependencies), but there is no problem with extending it for multiple edge labels.

5.2. Adapting MetaDex to graph model

The calculation of the cost values for vertex deletion and substitution can be done in an arbitrary way. Since we will need all the costs for further calculations anyway, we assume that we calculate them before running the GED, which can be used to reduce the time speed in case of a more complex comparison.

In this work, we propose an approach that focuses mainly on expression comparison. For each outer vertex defined in Section 2.2. we collect the expressions occurring in it and then flatten them into a vector using the bag-of-words model. For example, for the expression $"a + b \cdot c \textit{ where } b > d"$, where a, b, c, and d are values of some columns in some tables, we obtain the vector $\{"a" : 1, "b" : 2, "c" : 1, "d" : 1, " + " : 1, " \cdot " : 1, " > " : 1\}$. We do this separately for value and control expressions, and we keep information about what type of transformation it is. The cost of deletion is the sum of all values in the vector. When two transformations are of different types, we assume that the substitution cost is equal to the sum of deleting one and inserting the other. If the types are the same, the substitution cost is the edit distance for the vectors (the sum of the absolute values of the differences of all coordinates).

5.3. Data Cleaning

ETL processes defined in this way may have information that we do not want to compare. We consider processes defined at the lowest level, where they do not usually occur alone. They often have defined parameters that are specified in the wider scope, or provided by other procedures. Another example is parts of processes that, for example, are responsible for logging, which is not part of the process activity. Furthermore, the same operation can be performed in different ways, or at least in different order.

Therefore, we do two things. We remove all transformations that are not the source of any output table. The second thing is to standardize the most common transformations: filter, join and transformer (calculation). We require the filter to occur before the transformer if the calculation is independent of it. Furthermore, if we are filtering data from only one join source, we require the filtering to occur before the joining of the two sources.

6. Calculating the GED

ETL procedures are often small enough to calculate an exact GED. To calculate it, we use the search algorithm A*. It creates a path from the starting vertex, each time selecting a vertex x from the available unexplored vertices at a given step to minimize the function

$$f(x) = g(x) + h(x),$$

where $g(x)$ is a cost of mapped vertices, and $h(x)$ is a heuristic cost of unmapped vertices. To calculate h we use the Hungarian algorithm. The procedure is shown in algorithm 1. Very importantly, this heuristic always returns a lower bound on the true cost of matching, so that the A* algorithm always finds the optimal solution. The only difference is that in our case, the graphs are directed. We assume that edges directed in opposite directions are just different edges. We also decided that it is hard to argue that value and control edges, or edges directed in opposite directions, are similar, and the cost of substitution is equal to the cost of deleting one of them and inserting the other.

We propose two improvements to the above algorithm. The first is to sort the vertices in order to traverse neighbouring vertices and thus obtain the real cost associated with editing edges faster. Since it may be difficult to define such an order, here we benefit from the fact that the vast majority of ETL processes does not contain cycles. In this case, it is enough to sort the vertices topologically. The second proposal is to reduce the search tree in the A* algorithm. In most

[2]Formally p is a set of pairs of vertices, but sometimes it is more convenient to think that it contains vertices from each graph and a function, which maps vertices from the first graph to the second. This abuse of notation does not cause any serious problems.

Algorithm 1: A* – Algorithm for Calculating the Exact GED

```
   Input  : Graphs $\mathcal{G}_1 = (V_1, E_1)$, $\mathcal{G}_2 = (V_2, E_2)$, where $V_1 = \{u_1, \ldots, u_{n_1}\}$,
            $V_2 = \{v_1, \ldots, v_{n_2}\}$
1  Initialize $OPEN$ as an empty priority queue;
2  Insert $(u_1 \to w)$ to $OPEN$ for all $w \in V_2$;
3  Insert $(u_1 \to \epsilon)$ to $OPEN$;
4  while no solution is found do
5      Select and remove $p$ with minimum $(g(p) + h(p))$ from $OPEN$;
6      if p is a valid edit path then
7          return p as the solution;
8      else
9          Assume $p$ contains[2]$\{u_1, \ldots, u_k\} \subseteq V_1$ and $W \subseteq V_2$;
10         if $k \leq n_1$ then
11             Insert $p \cup (u_{k+1} \to v_i)$ to $OPEN$ for all $v_i \in V_2 \setminus W$;
12             Insert $p \cup (u_{k+1} \to \epsilon)$ to $OPEN$;
13         else
14             Insert $p \cup \bigcup_{v_i \in V_2 \setminus W} (\epsilon \to v_i)$ to $OPEN$;
   Output: An optimal edit path from $\mathcal{G}_1$ to $\mathcal{G}_2$.
```

applications of using GED to calculate the similarity of ETL processes, it will be much more important to obtain a similarity measure for processes that are similar. Taking advantage of the fact that these processes are acyclic, we can restrict to comparing vertices only with those that are in a similar place in the second graph. More formally, we perform the search in A* according to the topological order. For each vertex, we calculate distance from input and output tables. Then, we restrict the search to vertices whose distance from the input or output tables does not differ from a certain fixed value.

7. Experiments

7.1. Datasets

We used 3 datasets from the actual, real-world projects for our experiments: A, B, C[3]. The datasets contain 1500, 7500 and 14000 processes respectively, with an average number of 41, 45 and 33 vertices.

[3]**NDA Statement.** Due to NDA and intellectual property issues, the company whose data was used for the study did not agree to identify and describe it in detail. Therefore, the datasets (containing the sets of the ETL processes) used in this research are anonymized and described as datasets A, B and C.

7.2. Compared algorithms

We implemented the two most popular algorithms for solving the edit distance problem as our baseline, together with two algorithms proposed by us:

- **Hungarian** [35] – the traditional inexact solver based on Hungarian algorithm.
- **Hungarian-A*** [36] – an exact solver based on A* search algorithm using Hungarian algorithm as an estimation heuristic to guide search space exploration
- **Hungarian-A*-top** – our proposal; a version of Hungarian-A*, with traversing the vertices in topological order.
- **Hungarian-A*-cut** – a variant of our proposal; an approximation algorithm that reduces search space, forcing the matched vertices to be at a similar location in the graph.

7.3. Experiment setup

The datasets are too large to calculate a similarity on each pair. In view of this, we randomly selected 1000 pairs of graphs for each dataset, and we performed a comparison for them. However, randomly selected graphs tend to be dissimilar, and we are more interested to compare graphs that may be similar. To deal with this, we performed a clustering based on a flattened vector created by summing the vectors for all vertices. We then selected another 1000 pairs of graphs, comparing only graphs from the same cluster. Moreover, in practice, an exact GED value can only be calculated for small graphs, so for our experiments we considered graphs with at most 20 vertices for randomly selected graphs, and with at most 50 vertices for graphs selected from the same cluster. We set the cost of deleting the edges at 10, because small values do not have much impact on the GED value. The maximum difference in distances between the matched vertices for Hungarian-A*-cut was set to 2. For each pair of graphs we collected information about the optimal GED, the value obtained (in the case of the approximation algorithm), the runtime of the algorithm, and the size of the graphs. We also calculated the level of similarity between the two graphs as

$$sim(\mathcal{G}_1, \mathcal{G}_2) = 1 - \frac{GED(\mathcal{G}_1, \mathcal{G}_2)}{DEL(\mathcal{G}_1) + DEL(\mathcal{G}_2)},$$

where $DEL(\mathcal{G})$ is a cost of deleting the entire graph $\mathcal{G}$.

We used the `jgrapht` library [30] to perform the Hungarian algorithm.

Table 2.: Evaluation of all methods on datasets A, B and C

Method	Dataset A			Dataset B			Dataset C		
	cost	p	time	cost	p	time	cost	p	time
Randomly selected graphs									
Hungarian	319	0.11	**0.0008**	305	0.159	**0.0008**	393	0.11	**0.0008**
Hungarian-A*	**287**	-	5.41	**250**	-	2.16	**352**	-	5.62
Hungarian-A*-top	**287**	-	4.45	**250**	-	1.08	**352**	-	3.47
Hungarian-A*-cut	289	**0.004**	0.024	253	**0.007**	0.031	356	**0.009**	0.037
Graphs selected from the same cluster									
Hungarian	50.1	0.034	**0.0005**	111	0.10	**0.0005**	69.5	0.059	**0.0005**
Hungarian-A*	**3.18**	-	22.5	**4.57**	-	0.182	**4.85**	-	1.55
Hungarian-A*-top	**3.18**	-	0.28	**4.57**	-	0.095	**4.85**	-	0.75
Hungarian-A*-cut	**3.18**	**0**	0.17	**4.57**	**0**	0.039	5.49	**0.001**	0.06

7.4. Results

We compared the average calculated edit distance cost, the time (measured in seconds), and – for the approximation algorithms – the average error of approximation of the level of similarity between graphs, relative to the true similarity calculated by exact GED (denoted by p). The evaluation of given datasets for both randomly selected graphs and for graphs coming from the same cluster is shown in table 2.

The **Hungarian** algorithm performs one matching in polynomial time and is therefore unquestionably much faster than the others. However, it does not take into consideration how the vertices are related to each other. The experimental results may suggest that it nevertheless obtained a high similarity score, being wrong by several percentage points. This may be due to the fact that much of the similarity of the graphs is due to the similarity of the vertices rather than the edges.

The **Hungarian-A*-top** compared to benchmark **Hungarian-A*** performs significantly faster. On random graphs it is several tens of percent faster, up to about double for graphs that have a significant chance of being similar. The difference in performance of these algorithms increases for larger graphs and where the edit distance is higher. We can see this in the example of dataset A, for graphs coming from the same cluster, where there were more such pairs, and then the huge advantage of the Hungarian-A*-top algorithm can be seen there.

The **Hungarian-A*-cut** is an approximation algorithm that, compared to Hungarian-A* or Hungarian-A*-top, searches far fewer vertices and is many times faster. Still, it finds solutions close to the optimum, and very often finds the optimal graph edit distance. This can be seen especially in the examples of graphs that are selected from the same cluster, where in the case of two datasets Hungarian-A*-cut the optimal solutions were found for all examples. However, this is not very surprising when we compare to the Hungarian algorithm, which found decent solutions without any search.

This also shows the specificity of ETL processes. The vast majority of processes are either very similar or completely different. So if we pre-select the graphs that are likely to be similar, it turns out that even a strongly restricted search is usually enough to find the optimal graph edit distance.

8. Conclusions

Comparing ETL processes is a difficult task, while useful for better management of the data warehouse. In this paper, we have proposed to use a common graph model and to compare ETL processes using a method based on edit distance. For smaller processes it is possible to obtain an exact value, for larger ones one has to use approximation algorithms. Our proposed modifications have shown that it is possible to save the running time, taking advantage of the fact that ETL processes usually do not contain cyclic dependencies.

9. References

[1] Alexander Albrecht and Felix Naumann. Managing ETL Processes. In *International Workshop on New Trends in Information Integration*, 2008.

[2] Alexander Albrecht and Felix Naumann. Systematic ETL management - Experiences with high-level operators. In *MIT International Conference on Information Quality*, 2013.

[3] Michael Becker and Ralf Laue. A comparative survey of business process similarity measures. *Comput. Ind.*, 63:148–167, 2012.

[4] Neepa Biswas, Samiran Chattapadhyay, Gautam Mahapatra, Santanu Chatterjee, and Kartick Chandra Mondal. A New Approach for Conceptual Extraction-Transformation-Loading Process Modeling. *Int. J. Ambient Comput. Intell.*, 10:30–45, 2019.

[5] David B. Blumenthal. New Techniques for Graph Edit Distance Computation. *ArXiv*, abs/1908.00265, 2019.

[6] David B. Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. Comparing heuristics for graph edit distance computation. *The VLDB Journal*, 29:419–458, 2019.

[7] Sébastien Bougleux and Luc Brun. Linear Sum Assignment with Edition. *ArXiv*, abs/1603.04380, 2016.

[8] Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognit. Lett.*, 87:38–46, 2017.

[9] Sébastien Bougleux, Benoit Gaüzère, David B. Blumenthal, and Luc Brun. Fast linear sum assignment with error-correction and no cost constraints. *Pattern Recognit. Lett.*, 134:37–45, 2020.

[10] Sébastien Bougleux, Benoit Gaüzère, and Luc Brun. A Hungarian Algorithm for Error-Correcting Graph Matching. In *Workshop on Graph Based Representations in Pattern Recognition*, 2017.

[11] Silvio Cesare, Yang Xiang, and Wanlei Zhou. Control Flow-Based Malware VariantDetection. *IEEE Transactions on Dependable and Secure Computing*, 11(4):307–317, 2014.

[12] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12:41–58, 2003.

[13] Stefan Deßloch, Mauricio A. Hernández, Ryan Wisnesky, Ahmed M. Radwan, and Jindan Zhou. Orchid: Integrating Schema Mapping and ETL. *2008 IEEE 24th International Conference on Data Engineering*, pages 1307–1316, 2008.

[14] Asma Dhaouadi, Khadija Bousselmi, Mohamed Mohsen Gammoudi, Sébastien Monnet, and Slimane Hammoudi. Data Warehousing Process Modeling from Classical Approaches to New Trends: Main Features and Comparisons. *Data*, 7:113, 2022.

[15] Remco M. Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Graph Matching Algorithms for Business Process Model Similarity Search. In *International Conference on Business Process Management*, 2009.

[16] Remco M. Dijkman, Marlon Dumas, Boudewijn F. van Dongen, Reina Uba, and Jan Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36:498–516, 2011.

[17] Remco M. Dijkman, Boudewijn F. van Dongen, Marlon Dumas, Luciano García-Bañuelos, Matthias Kunze, Henrik Leopold, Jan Mendling, Reina Uba, Matthias Weidlich, Mathias Weske, and Zhiqiang Yan. A Short Survey on Process Model Similarity. In *Seminal Contributions to Information Systems Engineering*, 2013.

[18] Dawid Duda, Jeffrey T. Pascoe, Wojciech Matyjewicz, and Krzysztof Maziarz. Method and apparatus for analyzing and migrating data integration applications, U.S. Patent No. 2012/0296862 A1, Nov. 2012.

[19] Yang Fa. Malware Detection Based on Graph Edit Distance. *Journal of Wuhan University*, 2013.

[20] Lester Randolph Ford and Delbert Ray Fulkerson. Maximal Flow Through a Network. *Canadian Journal of Mathematics*, 8:399 – 404, 1956.

[21] Daniela Grigori, Juan Carlos González Corrales, Mokrane Bouzeghoub, and Ahmed Gater. Ranking BPEL Processes for Service Discovery. *IEEE Transactions on Services Computing*, 3:178–192, 2010.

[22] Informatica. Informatica Intelligent Cloud Services. https://www.informatica.com/products/cloud-integration.html.

[23] Md Rofiqul Islam and Tomas Cerny. Business Process Extraction Using Static Analysis. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1202–1204, 2021.

[24] ISO/IEC 25010 : 2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, 2011.

[25] Roy Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[26] Derek Justice and Alfred O. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1200–1214, 2006.

[27] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52, 1955.

[28] Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, feb 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[29] Chih-Long Lin. Hardness of Approximating Graph Transformation Problem. In *International Symposium on Algorithms and Computation*, 1994.

[30] Dimitrios Michail, Joris Kinable, Barak Naveh, and John V. Sichi. JGraphT—A Java Library for Graph Data Structures and Algorithms. *ACM Trans. Math. Softw.*, 46(2), May 2020.

[31] James R. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of The Society for Industrial and Applied Mathematics*, 10:196–210, 1957.

[32] Bruno Oliveira and Orlando Belo. ETL Standard Processes Modelling - A Novel BPMN Approach. In *International Conference on Enterprise Information Systems*, 2013.

[33] Basel Committee on Banking Supervision. Principles for effective risk data aggregation and risk reporting, 2013.

[34] Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *J. Graph Theory*, 1:339–363, 1977.

[35] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27:950–959, 2009.

[36] Kaspar Riesen, Stefan Fankhauser, and Horst Bunke. Speeding Up Graph Edit Distance Computation with a Bipartite Heuristic. In *Mining and Learning with Graphs*, 2007.

[37] Elena Samota. Representing ETL Flows with BPMN 2.0, 2015.

[38] Francesc Serratosa. Fast computation of Bipartite graph matching. *Pattern Recognit. Lett.*, 45:244–250, 2014.

[39] Alkis Simitsis. Modeling and managing ETL processes. In *VLDB PhD Workshop*, 2003.

[40] Alkis Simitsis. Mapping conceptual to logical models for ETL processes. In *International Workshop on Data Warehousing and OLAP*, 2005.

[41] Alkis Simitsis, Panos Vassiliadis, Umeshwar Dayal, Anastasios Karagiannis, and Vasiliki Tziovara. Benchmarking etl workflows. In Raghunath Nambiar and Meikel Poess, editors, *Performance Evaluation and Benchmarking*, volume 5895, pages 199–220, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[42] Alkis Simitsis, Panos Vassiliadis, Manolis Terrovitis, and Spiros Skiadopoulos. Graph-Based Modeling of ETL Activities with Multi-level Transformations and Updates. In *International Conference on Data Warehousing and Knowledge Discovery*, 2005.

[43] Juan Trujillo and Sergio Luján-Mora. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In *International Conference on Conceptual Modeling*, 2003.

[44] Kevin Wilkinson, Alkis Simitsis, Malú Castellanos, and Umeshwar Dayal. Leveraging Business Process Models for ETL Design. In *International Conference on Conceptual Modeling*, 2010.

[45] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing Stars: On Approximating Graph Edit Distance. *Proc. VLDB Endow.*, 2:25–36, 2009.