Detecting Duplicate Records in Scientific Workflow Results

Khalid Belhajjame[1], Paolo Missier[2] and Carole A. Goble[1]

[1]School of Computer Science
University of Manchester
Oxford Road, Manchester, UK
{Khalid.Belhajjame,Carole.Goble}@cs.man.ac.uk
[2]School of Computing Science,
Newcastle University, Newcastle, U.K.
Paolo.Missier@ncl.ac.uk

Abstract. Scientific workflows are often data intensive. The data sets obtained by enacting scientific workflows have several applications, e.g., they can be used to identify data correlations or to understand phenomena, and therefore are worth storing in repositories for future analyzes. Our experience suggests that such datasets often contain duplicate records. Indeed, scientists tend to enact the same workflow multiple times using the same or overlapping datasets, which gives rise to duplicates in workflow results. The presence of duplicates may increase the complexity of workflow results interpretation and analyzes. Moreover, it unnecessarily increases the size of datasets within workflow results repositories. In this paper, we present an approach whereby duplicates detection is guided by workflow provenance trace. The hypothesis that we explore and exploit is that the operations that compose a workflow are likely to produce the same (or overlapping) dataset given the same (or overlapping) dataset. A preliminary analytic and empirical validation shows the effectiveness and applicability of the method proposed.

1 Introduction

Scientific workflows are increasingly used by scientists as a means for specifying and enacting their experiments. Such workflows are often data intensive [5]. The data sets obtained by their enactment have several applications, e.g., they can be used to understand new phenomena or confirm known facts, and therefore are worth storing (preserving) for future analyzes. For example, such datasets can be stored in public repositories, and made available from within the linked data cloud [17], to be browsed, queried, analyzed and used to feed the execution of other workflows.

Because of the exploratory nature of research investigations, the datasets obtained by workflow executions often contain duplicate data records. (By record, we mean an instance that is used to feed an input parameter or is generated by an output parameter.) Indeed, scientists tend to enact the same workflow several times using the same or overlapping datasets, which gives rise to duplicates

in workflow results. Typically, the duplicate records generated as a result are assigned different identifiers by the workflow engine. This yields the following undesirable outcomes: *i)*- The analysis and interpretation of workflow results may become cumbersome and tedious, as it is up to the scientist to identify the data records that are semantically identical, to eventually draw scientific conclusions. *ii)*- Moreover, the presence of duplicate records unnecessarily increases the size of datasets within workflow results repositories.

Existing record linkage techniques [8] can be used to detect duplicates in workflow results. Consider a workflow wf that has been enacted for many times over a given period of time, and that the results of the workflow and its constituent operations were stored. To identify duplicates that were used or generated by the operations that compose the workflow wf, record linkage techniques can be applied to those records. Specifically, given the set of records R_i (resp. R_o) that were used (resp. generated) by a constituent operation *op* of wf, the records in R_i (resp. R_o) are compared to detect duplicate records. Comparing all possible pairs to identify duplicate records in a set R can be expensive when R is large: the number of record pair comparisons grows quadratically with the number of records to be matched.

To overcome the above problem, a number of researchers have investigated the use of blocking methods [1] to reduce the number of record pair comparisons. The underlying idea of blocking methods is to split the set of records to be compared into subsets, known as *blocks*. Two records are compared only if they belong to the same block. While their effectiveness have been proven, blocking methods are highly domain dependent. They require some detective work from the part of a domain expert who identifies the subset of attributes that can be used for forming blocks or provides training data that can be used to learn blocking criteria [13, 2].

In this paper, we explore and exploit an additional and different source of information, namely provenance traces collected when enacting workflows, to guide the detection of duplicates in workflow results. Specifically, we make the following contributions:

- **A method for guiding duplicates detection in workflow results.** Rather than comparing pairwise the data records bound to every operation parameter within a workflow, we show how the results of record pair comparisons can be reduced to a subset of operation parameters based on provenance trace.
- **Extension of the method proposed to support collection-based workflows.** We show how the method proposed can be extended to support duplicate detection in the context of collection-based workflows in which operation parameters can use and/or generate a set of records within a single operation invocation.
- **Validation of the method proposed**. We report on the results of an analytical and empirical validation that shows the effectiveness and applicability of the method proposed.

The paper is structured as follows. We begin by analyzing and comparing related work to ours (in Section 2). We present the model that we use, for the purposes of this paper, to define scientific workflows and the provenance trace obtained by their enactment (in Section 3). We then present the algorithm that we propose for detecting duplicate records in workflow results (in Section 4). We report on the results of a preliminary evaluation (in Section 5), and close the paper discussing our ongoing and future work (in Section 6).

2 Related Work

Research in duplicate record detection has been active for more than three decades. Elmagarmid *et al.* [8] conducted a thorough analysis of the literature in this field. They covered the similarity metrics used for matching individual record fields, the techniques for comparing records, as well as the systems providing such capabilities.

As mentioned earlier, the number of record pair comparisons grows quadratically with the number of records to be matched $O(n^2)$. To improve the efficiency of duplicate detection, several blocking techniques [1] have been devised. Using such techniques, the set of records to be compared is subdivided into a set of mutually exclusive blocks. Two records are compared only if they belong to the same block. Typically, blocking techniques reduces the number of record pair comparisons to $O(\frac{n^2}{b})$, where b is to the number of blocks [6]. As well as blocking, other techniques have been proposed to improve the efficiency of duplicate detection. Using the Sorted Neighbourhood, for example, the records are sorted based on a sorting key. Two records are then compared only if they are within a window of a fixed size w. As a result, the total number of record comparisons using the Sorted Neighbourhood is $O(wn)$ [1].

The above techniques require user inputs. For example, to use blocking techniques, the user, or domain expert, needs to identify the attributes that can be used to split the set of records into blocks. Often, this is a trail-and-error task [16], in which the user examines records attributes, and select the ones that will (or are expected to) yield *good* partitioning of the set of records.

To reduce the complexity of this task, some researchers investigated the use of machine learning techniques [15, 9, 11]. Generally speaking, using such techniques, the attributes to be used in blocking are selected based on training data, which take the form of records that are known to be duplicates and other that are known not to be duplicates, and which are provided by the domain expert.

As well as supervised machine learning techniques, some researchers have investigated the use of unsupervised machine learning techniques for record linkage. For example, Michalowski *et al.* [10] showed how duplicates can be identified by using secondary sources such as location, phone number, etc. Elfeky *et al.* [7] proposed an algorithm that combines both supervised and unsupervised machine learning techniques to detect duplicate records. Specifically, they use a two-step process whereby record classes are first identified using clustering, then super-

vised machine learning techniques are applied to classify the records within the classes identified.

The method that we present in this paper is not an alternative to the above techniques. Rather it is complementary: it is meant to further improve the efficiency of the above duplicate detection methods in the context of workflow results by exploiting provenance traces to propagate the results of record pair comparison along the operations parameters that are connected within the workflow.

The method we present in this paper can also be useful when the number of records to be compared is small, as it reduces the need for data preparation [8] to few operation parameters within the workflow. Indeed, the (raw) records instances of a given operation parameter are often long complex strings. Consider for example the *SearchSimple* service operation provided by the DNA Data Bank of Japan[1]. The records used as input to such operation are biological entries, which takes the form of long strings containing complex information specifying the accession of the biological entry, its accession number, organism, motif, cross-references to biological data sources, etc. Moreover, such entries may be formatted using different representations, e.g., Uniprot, Fasta, IPR. Therefore, comparing such records based on their textual content may lead to detecting false duplicates and missing true ones. For example, two records that represent the same biological entry may be found to be different because they are formatted using different representations. On the other hand, two different records may be found to be identical because they have similar content. To avoid the above issues, duplicate detection is often preceded by a data preparation phase stage [8] during which the raw records are parsed to identify individual data elements and then transformed into structured, uniformly formatted, and therefore comparable, records. Since the parameters of the operations that compose a workflow can be (and are typically) semantically and syntactically different, data preparation may turn out to be expensive as it potentially requires building a parser for every operation parameter. The method that we describe in this paper eases this problem, since data preparation is required only for a subset of operation parameters within the workflow.

3 Data-Driven Workflows and Provenance Trace

We focus in this paper on the problem of identifying duplicate records that are used or generated by data-driven workflows. A data driven workflow is a directed acyclic graph $wf = \langle N, E \rangle$. A node $\langle op, I_{op}, O_{op} \rangle \in N$ represents an analysis operation op, which can be implemented as a Java program, a Perl script or provided by a third party web service, has a set of ordered input parameters I_{op}, and has a set of ordered output parameters O_{op}. The edges are data flow dependencies specifying how the data records generated by a given operation are used by the succeeding operation(s) within the workflow. Therefore, an edge

[1] www.ddbj.nig.ac.jp

$\langle\langle op, o\rangle, \langle op', i\rangle\rangle \in E$ is a pair that connects the output o of the op operation to the input i' of another operation op'.

The execution of workflows gives rise to provenance trace, which we capture using two relations: transformation and transfer [12]. Consider an operation op that has n input parameters $I_{op} = \langle i_1, \ldots, i_n\rangle$, and m output parameters $O_{op} = \langle o_1, \ldots, o_m\rangle$, we use

$$\langle op, \langle o_1, r_{o_1}\rangle\rangle, \ldots, \langle op, \langle o_m, r_{o_m}\rangle\rangle \leftsquigarrow \langle op, \langle i_1, r_{i_1}\rangle\rangle, \ldots, \langle op, \langle i_n, r_{i_n}\rangle\rangle \quad (1)$$

to denote the transformation relation specifying that the execution of the op operation within a workflow took as input the ordered set of records $\langle r_{i_1}, \ldots, r_{i_n}\rangle$, and generated the ordered set of records $\langle r_{o_1}, \ldots, r_{o_m}\rangle$, where r_{x_i} denotes a record that is instance of the input or output parameter x_i. For exposition sake, we use in what follows $OutB_{op} \leftsquigarrow InB_{op}$ to denote the transformation relation in (1), where InB_{op} denotes the set of input bindings $\langle op, \langle i_1, r_{i_1}\rangle\rangle, \ldots, \langle op, \langle i_n, r_{i_n}\rangle\rangle$, and $OutB_{op}$ denotes the set of output bindings $\langle op, \langle o_1, r_{o_1}\rangle\rangle, \ldots, \langle op, \langle o_m, r_{o_m}\rangle\rangle$.

As well as transformation relations connecting output records to input records of an operation execution, provenance trace also caters for transfer relations which specify transfer of records along the edges of the workflow between different operations. Specifically, we use:

$$\langle op', \langle i', r\rangle\rangle \leftarrow \langle op, \langle o, r\rangle\rangle \quad (2)$$

to denote the transfer relation specifying that the record r generated by the output parameter o of the operation op was used to feed the input i' of the operation op'.

Together, the transformation and transfer relations defined above, are used to encode provenance trace $\mathcal{T}$ obtained by the execution of workflows.

4 Provenance-Guided Detection of Duplicates

In this section, we present a method for identifying duplicate records in workflow results.

To guide the detection of duplicates, we exploit the following observation. Consider op an operation that is used within a workflow, and consider that i and o are respectively an input parameter and output parameter of op. If the operation op is known to be deterministic, then two records r and r' instances of the output o are identical if they are generated using the same set of input bindings, i.e.:

$$\begin{aligned} deterministic(op) \wedge (\exists\ (OutB_{op} \leftsquigarrow InB_{op} \in \mathcal{T})\ \wedge\ (OutB'_{op} \leftsquigarrow InB_{op} \in \mathcal{T}) \\ s.t.\ (\langle op, \langle o, r\rangle\rangle \in OutB_{op})\ \wedge\ (\langle op, \langle o, r'\rangle\rangle \in OutB'_{op})) \\ \Rightarrow id(r, r') \end{aligned} \quad (3)$$

$id(r, r')$ denotes that two records r and r' are identical.

If the operation op is known to be injective, then two records r and r' that are instances of the input i are identical if they yield the same set of output bindings, i.e.:

$$injective(op) \wedge (\exists\ (OutB_{op} \leftsquigarrow InB_{op} \in \mathcal{T})\ \wedge\ (OutB_{op} \leftsquigarrow InB'_{op} \in \mathcal{T}) \\ s.t.\ (\langle op, \langle i, r\rangle\rangle \in InB_{op})\ \wedge\ (\langle op, \langle i, r'\rangle\rangle \in InB'_{op})) \\ \Rightarrow id(r, r') \qquad (4)$$

The above rules can be used to substantially reduce the number of records that need to be compared for detecting duplicates in workflow results. In particular, if the operations that compose the workflows are known to be deterministic, then the records used as input to the workflow as a whole, i.e., those used to feed the starting operation(s) within the workflow, can be compared. Rule 3 can then be applied transitively to identify duplicates generated by other operations in the workflow. On the other hand, if the operations that compose the workflow are known to be injective, then the records generated by the workflow as a whole, i.e., those generated by the last operation(s) w.r.t. to the dataflow, can be compared. Rule 4 can then be applied transitively to records that are used as input to the operations within the workflow.

The method that we present in this paper for detecting duplicates assumes that operations are deterministic. In other words, workflow containing non deterministic operations are outside the scope of this paper. Generally, the operations that constitute a workflow may not be deterministic. It is nevertheless important to study the special case where operation determinism holds, especially that the empirical evaluation that we will report on in Section 5 suggests that most analysis operations are deterministic. Note, also, that we will present, later on in Section 4.1, a technique that can be used to check if a given analysis operation is deterministic, and therefore can be used to identify the workflows on which the method we present in this section can be safely applied.

Given the above discussion, we present in what follow an algorithm in which operations are assumed to be deterministic. The algorithm for detecting duplicates operates as illustrated in Figure 1. In what follow, we present in details the phases outlines in Figure 1.

Phase 1. Given a workflow wf and the provenance trace $\mathcal{T}$ obtained by executing the workflow wf multiple times, in the first phase the records that are bound to the input parameters of each of the starting operations are compared to identify duplicate records. To illustrate this, consider that op_s is a starting operation of the workflow wf, i.e., the input parameters of op_s are not associated with any data links within the workflow wf, consider that i is an input of op_s, and consider that $R_i^{op_s}$ is the set of records that are bound with the input i in the provenance trace $\mathcal{T}$. In the first phase, we compare the records in $R_i^{op_s}$ to identify duplicate records. The techniques used for matching the records are outside the scope of this paper. Matching techniques such as those provided by the Tailor

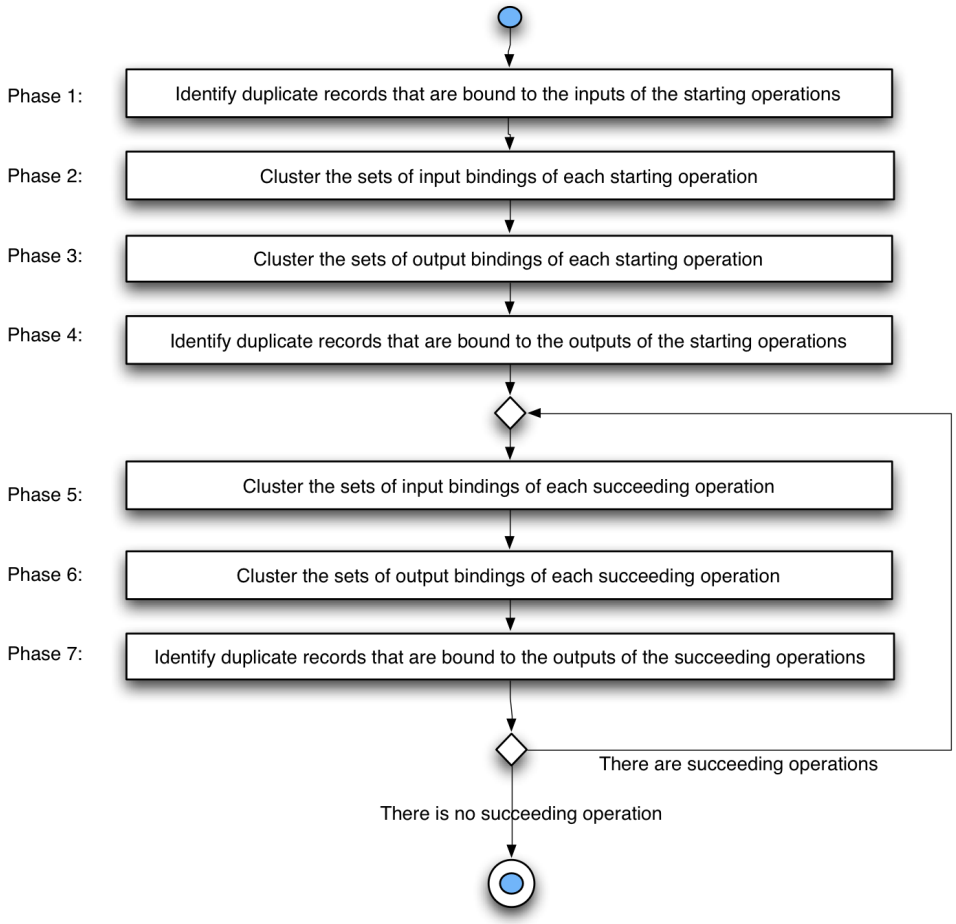

Fig. 1. Process for provenance-guided duplicates detection.

[6] and Febrl [3] systems can be used for this purpose. The result of this phase is a partition of disjoint sets $R_i^{op_s} = R_1 \cup \cdots \cup R_n$ where R_i, $1 \leqslant i \leqslant n$ is a set of duplicate records.

Phase 2. The sets of input bindings that are used to feed each starting operation of the workflow wf in the provenance trace $\mathcal{T}$ are compared and clustered into groups of identical sets of input bindings. To illustrate this, consider that the starting operation op_s has the following input parameters $I_{op_s} = \{i_1, \ldots, i_n\}$, and consider that $\mathcal{IB}_{op_s}$ is the sets of input bindings that are associated with the operation op_s in the provenance trace $\mathcal{T}$. Two sets of input bindings

$\langle op_s, \langle i_1, r_{i_1} \rangle \rangle, \ldots, \langle op_s, \langle i_n, r_{i_n} \rangle \rangle$ and $\langle op_s, \langle i_1, r'_{i_1} \rangle \rangle, \ldots, \langle op_s, \langle i_n, r'_{i_n} \rangle \rangle$ in $\mathcal{IB}_{op_s}$ are identical iff:

$$\forall\ 1 \leqslant k \leqslant n,\ id(r_{i_k}, r'_{i_k})$$

After comparing the sets of input bindings in $\mathcal{IB}_{op_s}$, they are clustered into groups of identical sets of bindings. For example, consider that $\mathcal{IB}_{op_s}$ contains the following 5 sets of input bindings: $\mathcal{IB}_{op_s} = \{InB_1, \ldots, InB_5\}$. The following clustering $\{\{InB_1, InB_2, InB_4\}, \{InB_2, InB_5\}\}$ specifies that the bindings InB_1, InB_3 and InB_4 are identical, and that the bindings InB_2 and InB_5 are identical.

Phase 3. Given the clustering of the input bindings of the starting operation op_s of the workflow wf, provenance trace $\mathcal{T}$ is used to cluster the sets of outputs bindings of op_s into groups of identical sets of output bindings. To do so, we exploit the clustering of sets of input binding obtained in *phase 1*. Specifically, two sets of output bindings $OutB_{op}$ and $OutB'_{op}$ are identical, and therefore clustered together if they are obtained using identical input bindings. In other words, two sets of output binding $OutB_{op_s}$ and $OutB'_{op_s}$ of the operation op_s are identical, and therefore grouped into the same cluster, if:

$$\exists\ InB_{op_s}, InB'_{op_s} \in \mathcal{T}$$
$$s.t.\ OutB_{op} \leftsquigarrow InB_{op}\ \wedge\ OutB'_{op} \leftsquigarrow InB'_{op}\ \wedge\ id(InB_{op_s}, InB'_{op_s})$$

where $id(InB_{op_s}, InB'_{op_s})$ denotes that the sets of bindings InB_{op_s} and InB'_{op_s} are identical.

Phase 4. In this phase, the records that are bound to the output parameters of each starting operation op_s are identified given the clustering obtained in *phase 3*. As an example, consider two sets of output bindings of the operation op_s: $OutB_{op_s} = \langle op_s, \langle o_1, r_{o_1} \rangle \rangle, \ldots, \langle op_s, \langle o_m, r_{o_m} \rangle \rangle$ and $OutB'_{op_s} = \langle op_s, \langle o_1, r'_{o_1} \rangle \rangle, \ldots, \langle op_s, \langle o_m, r'_{o_m} \rangle \rangle$. If $OutB_{op_s}$ and $OutB_{op_s}$ are in the same group according to the clustering obtained in *phase 3*, then the records r_{o_i} and r'_{o_i} are identical for $1 \leqslant i \leqslant m$.

Phase 5. The sets of input bindings that are associated with each operation op that succeeds the starting operations in the workflow wf are clustered into groups of identical sets of input bindings. To illustrate this, consider that $InB_{op} = \langle op, \langle i_1, r_{i_1} \rangle \rangle, \ldots, \langle op, \langle i_n, r_{i_n} \rangle \rangle$ and $InB'_{op} = \langle op, \langle i_1, r'_{i_1} \rangle \rangle, \ldots, \langle op, \langle i_n, r'_{i_n} \rangle \rangle$ are two sets of input bindings of the operation op. InB_{op} and InB'_{op} are identical, and therefore grouped into the same cluster, if the records r_{i_k} and r'_{i_k}, $1 \leqslant k \leqslant n$ are identical, i.e., $\forall\ 1 \leqslant k \leqslant n,\ id(r_{i_k}, r'_{i_k})$.

Phase 6. Just like with the starting operation op_s, the sets of output bindings associated with each of the operations that succeed the operation op_s are clustered into groups of identical sets of bindings (see *phase 3*).

Phase 7. The set R_o^{op} of records that are bound to each output parameter o of a succeeding operation op are partitioned into disjoint sets of identical records. This phase is similar to *phase 4*.

Phases $5, 6$ and 7 are repeated until treating, i.e., identifying duplicates, in records that are bound to the output parameters of each of the termination operations in the workflow wf. By termination operations, we mean operations with output parameters that are not associated with any data links in the workflow wf.

In the above algorithm, we make use of rule 3 (in *phase 3* and *phase 6*), which assumes that operations are deterministic. We can also identify duplicates by using rule 4 instead, which can be used when the operations that compose the workflow wf are known to be injective. To do so, the algorithm presented above needs to be modified. Specifically, the algorithm starts by comparing the records that are produced by the outputs of the termination operations within the workflow. Then using transitively rule 4, we identify the records that are bound to the remaining operation parameters in the workflow.

4.1 Verifying The Determinism of Analysis Operations

As mentioned earlier, the algorithm presented assumes that the operations that compose the workflow are known to be deterministic. If the source code of analysis operations is available, then program analysis techniques [4] can be employed to verify whether they are deterministic. In practice, analysis operations that are supplied by third parties often come without source code. For those operations, we can use the following approach to check whether they are deterministic. Given an operation op, we select examples values that can be used by the inputs of op, and invoke op using those values multiple times. We then examine the values produced by the operation. If the operation produces identical output values given identical input values, then it is likely to be deterministic, otherwise, it is not deterministic. Note that we say *likely to be deterministic*, since an operation may, in certain corner cases, be deterministic for the examples we selected but not for the whole space of legal input values. Note that such tests should be performed continuously over time. Indeed, as we shall explain later on in Section 5.2, many analysis operations use underlying data sources in their computation, and, as a result, updates to those sources may break the determinism of those operations. Therefore, tests performed for checking the determinism of operations should be performed over time to determine the window of time during which the operations in a given workflow remain deterministic.

4.2 Collection-Based Workflows

So far we have considered workflows in which operations take as input an ordered set of records each instance of a given input parameter and produce a set of ordered records each is an instance of a given output parameter. In practice, however, an important class of scientific workflows are collection-based workflows [12]. The analysis operations that constitute such workflows can have inputs

(resp. outputs) that consume (resp. generate) a set of records instead of a single record within a single operation invocation.

The algorithm presented in Section 4 needs to be slightly modified to be able to cater for collection-based workflows. In particular, we need to be able to identify when two sets are identical *(phase 1)*, and to identify duplicates records between two sets that are known to be identical *(phases 4,7)*. To illustrate this consider an operation *op* with an input i that takes a set of records. Given two sets of records R_i and R_j that are bound to the operation *op* in two different invocations within the provenance trace $\mathcal{T}$, we need to determine whether the two sets R_i and R_j are identical. To do so, we need to compare the records in R_i to the records in R_j. The sets R_i and R_j are identical if they are of the same size, and there is a bijective mapping $map : R_i \rightarrow R_j$ that maps each record r_i in R_i to a record r_j in R_j such that r_i and r_j are identical, i.e., $id(r_i, map(r_j))$.

Inversely, in *phases 4* and *7*, given two sets R_i and R_j that are known to be identical, we need to compare the records in R_i with the records in R_j to find a bijective mapping $map : R_i \rightarrow R_j$ that maps each record r_i in R_i to an identical record r_j in R_j.

5 Validation

To assess the effectiveness of the method presented in this paper, we performed two kinds of validation: analytical and empirical.

5.1 What is the benefit in terms of reducing the number of record pair comparisons?

We performed an an analytical analysis to understand the benefit that the method we described in this paper presents in terms of reducing the search space that needs to be explored to detect duplicate records. Consider a workflow wf, the operations of which are known to be deterministic. For simplicity sake, and without loss of generality, consider that the operations of wf have one input and one output, and that they are connected in sequence using data links. Let $\mathcal{T}$ be the provenance trace obtained by multiple executions of the workflow wf, and consider that n is the number of records that are bound to the input i of the starting operation op_s in the provenance trace $\mathcal{T}$. Consider that the workflow is composed of n_{op} operations. The number of record pair comparisons needed without using provenance trace is $(n_{op} + 1) \times N$, where N is the number of of record pair comparisons needed for a single operation parameter. For example, if the workflow is composed of two operations op_1 and op_2 that are connected in sequence, then we need $3 \times N$ record pair comparisons: N for comparing the records bound to the input of op_1, N for comparing the records bound to the output of op_1 and N for comparing the records bound to the output of op_2. Note that we do not need to compare the records bound to the input of op_2, since this input is connected to the output of op_2 by a data link, and therefore the set of

records bound to the input of op_2 is the same set of records bound to the output of op_1.

Using the method we described in this paper, we need to identify duplicates only for the starting operations in the workflow. In other words, the number of record pair comparisons is N. Using blocking techniques N is $\frac{n^2}{b}$, where b denotes the number of blocks. Notice that that number does not depend on the number of operations that compose the workflow. To illustrate the benefit our method can provide, consider the case in which the workflow is composed of 10 operations that have one input and one output, and are connected in sequence. And consider that 100 records are bound to each operation parameter, and that blocking techniques split the records associated with each operation parameter to 5 blocks. Using blocking techniques, without relying on provenance trace, requires 22000 record pair comparisons. This number is reduced to 2000 using the method presented in this paper. Notice that, the greater the number of operations that compose the workflow, the greater the reduction in terms of number of record pair comparisons.

5.2 Are real-world analysis operations deterministic?

The method we presented in this paper relies on the assumption that the operations that compose the workflow are deterministic. To have an insight on the degree to which this assumption holds in practice, we run an experiment using real world scientific workflows from the myExperiment repository [14]. Specifically, we selected 15 bioinformatics workflows that cover a wide range of analyzes, namely biological pathway analysis, sequence alignment, molecular interaction analysis. (Note that the myExperiment repository contains a large number of workflows, however, most of the workflows cannot be enacted for several reasons, notably the unavailability of the services that compose the workflows.) Together, the workflows we selected are composed of 151 operations. To identify which of these operations are deterministic, we run each of them 3 times using example values that were found either within myExperiment or Biocatalogue [**?**]. We then manually analyzed the output values of each operation. This analysis revealed that a small number of operations, namely 5 of 151 are not deterministic. After examining these 5 operations, it transpires they output URLs of files that contain the actual results of the computation. Note that although the URLs of the files generated by such operations were different between runs, the contents of the files were the same. The remaining operations, i.e., 146, generated the same output given the same input in the 3 invocations, and therefore are likely to be deterministic. We say *likely to be deterministic*, since an operation may, be deterministic for the examples we selected but not for the whole space of legal input values.

The results of the above experiment are encouraging, as it implies a broad applicability of the method described in this paper for propagating record pair comparison results. Note, however, that many of the operations that we analyzed access and use underlying data sources in their computation. For example, operations that perform sequence alignment use underlying sequences data sources.

Therefore updates to such sources may break the determinism assumption. This suggests that the determinism holds within a window of time during which the underlying sources remain the same, and that there is a need for monitoring techniques to identify such windows.

6 Conclusions and Future Work

The presence of duplicates in workflow results can hinder the analysis of the results, specially when the number of workflow executions is large. In this paper, we described a method that can be used to reduce the number of record pair comparisons and the need for data preparation to a subset of the parameters within the workflow. Preliminary validation of the proposed method is encouraging.

Our ongoing and future work includes further evaluation. As mentioned in the previous section, operation determinism may break because of updates to underlying data sources. In this respect, we are investigating new techniques for monitoring the determinism of analysis operations over time using test suites designed for this purpose. The monitoring results can be used to identify the cases in which the method described in this paper can be safely applied. We are also investigating ways to deal with the issue of false matches propagation. If two different records are identified as duplicates, then this may lead to detecting false matches using provenance trace. The same observation applies to false negatives propagation. If two identical records r and r' are not detected, then using provenance trace, we will fail in detecting identical records generated using r and r'. Note also that some true matches may not be identified using the method we described. In particular, a deterministic, yet not injective, operation within the workflow may output identical records given different input bindings. Using the algorithm described in Section 4 will not allow detecting those duplicates. We intend to conduct further evaluation to assess the scale at which false matches are propagated and true matches are missed. We are also investigating ways whereby our method can be adapted to alleviate the above issues, e.g., by running our method multiple times, not only once. Each time different parameters, not only the inputs of the starting operations, are selected as a starting point, and then cross-validating duplicate detection results obtained by the different runs. As well as the above, we are investigating ways in which the method presented can be adapted to identify duplicates across workflows, and conducting a user study to assess the usefulness of the method in practice.

Acknowledgment

We would like to thank the anonymous reviewers for their detailed and constructive comments.

References

1. Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of the KDD-2003 Workshop on Data*

Cleaning, Record Linkage, and Object Consolidation, pages 25–27, Washington, DC, 2003.
2. Mikhail Bilenko, Beena Kamath, and Raymond J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96. IEEE Computer Society, 2006.
3. Peter Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *KDD*, pages 1065–1068. ACM, 2008.
4. Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '79, pages 269–282, New York, NY, USA, 1979. ACM.
5. Ewa Deelman and Ann L. Chervenak. Data management challenges of data-intensive scientific workflows. In *CCGRID*, pages 687–692. IEEE Computer Society, 2008.
6. Mohamed G. Elfeky, Ahmed K. Elmagarmid, and Vassilios S. Verykios. Tailor: A record linkage tool box. In *ICDE*, pages 17–28. IEEE Computer Society, 2002.
7. Mohamed G. Elfeky, Thanaa M. Ghanem, Vassilios S. Verykios, Ahmed R. Huwait, and Ahmed K. Elmagarmid. Record linkage: A machine learning approach, a toolbox, and a digital government web service, 2003.
8. Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.
9. Mauricio A. Hernández and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2(1):9–37, 1998.
10. M. Michalowski, S. Thakkar, and C. Knoblock. Exploiting secondary sources for automatic object consolidation. In *Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 34–36, Washington, DC, 2003.
11. Matthew Michelson and Craig A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*. AAAI Press, 2006.
12. Paolo Missier, Norman W. Paton, and Khalid Belhajjame. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *EDBT*, pages 299–310. ACM, 2010.
13. Parag and Pedro Domingos. Multi-relational record linkage. In *Proceedings of the KDD-2004 Workshop on*, pages 31–48, August 2004.
14. David De Roure, Carole A. Goble, and Robert Stevens. The design and realisation of the myExperiment virtual research environment for social sharing of workflows. *Future Generation Comp. Syst.*, 25(5):561–567, 2009.
15. Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278. ACM, 2002.
16. W. E. Winkler. Approximate string comparator search strategies for very large administrative lists. Technical report, Statistical Research Report Series, US Census Bureau, 2005.
17. Jun Zhao, Satya Sanket Sahoo, Paolo Missier, Amit P. Sheth, and Carole A. Goble. Extending semantic provenance into the web of data. *IEEE Internet Computing*, 15(1):40–48, 2011.