A Difference Calculus for Structured Data

Jia Lin[†], Fiona Lawless and Christian Horn[*]

Department of Computing and Mathematics
Dundalk Institute of Technology, Dundalk, Co. Louth, Ireland

E-mail: [†]jia.lin@dkit.ie [*]christian.horn@dkit.ie

Abstract — **Traditional software configuration management tools employ existing Diff tools to identify the difference between intermediate versions of a document such as a source code file. When any new data structure arises, developers have to create a version control tool for this specific structure. This paper introduces a universal approach to the creation of a version control tool for artifacts of any structured data. The approach taken is to develop a difference calculus for arbitrary objects based on their types. This calculus is built on a type system in which types are nested. By constructing types recursively, more structured data can be modelled and the difference between successive versions of objects can be identified.**

Keywords — **Software Configuration Management, Version Control, Diff, Type System, Similarity Measure.**

I Introduction

Software Configuration Management (SCM) systems are used to store and track the changes made to intermediate versions of general text documents (in particular source code [1, 2]) which arise during the development of a software application, and to identify, patch and/or merge the differences between these versions[3].

Traditionally, SCM systems rely on existing diff tools which are limited in their scope [4], for example, the Revision Control System (RCS) [5] is built on the UNIX diff utility. Some other diff tools have been developed for other common data structures, such as XML files [6] or spreadsheet files [7, 8]. However, regardless of the type of the object being updated, changes take the form of a deletion and/or an insertion. Therefore, a generic version control tool could be developed to identify the difference between objects of arbitrary data structure.

Previous work has been done to detect changes between hierarchically structured data by representing the source files as trees. [9] developed an application, namely *LaDiff*, to identify and highlight changes between two LaTeX documents by converting the input files to ordered trees. [10] extended the above algorithm to detect changes between unordered trees and therefore allow the comparison of more data structures. The addition of a new "copy" operation in the edit set facilitates detection of more meaningful changes. Lempsink et al., in their paper "Type-safe diff for families of data types" [11], implemented a generic diff and patch using a functional programming language in which they defined a Diff type as a set of edit operations. The investigation started by applying the diff and patch algorithms to lists and trees respectively, a generic diff algorithm based on converting any data structure to a tree was then developed.

Our approach is different, instead of converting structured data to a tree strucure, we create a difference calculus based on a type system $\mathcal{T}$. Types in this type system have different structures and they are constructed recursively. The difference between objects of the same type is determined by the changes to the objects' components.

In this paper, the difference calculus and the type system $\mathcal{T}$ are described in section II. The difference calculus is composed of types in this type system and operators for these types. The similarity measure operator which is used to determine the similarity between two objects of the same type in this type system $\mathcal{T}$ is defined and discussed in section III. The directed difference

operator which is defined to identify the difference between two objects of the same type is presented in section IV. Section V presents an application by applying the difference calculus to graphs of maps. Finally, section VI is the conclusion.

II Difference Calculus

a) Type System

The investigation of our approach starts with the construction of a type system $\mathcal{T}$ containing primitive and structured types. Structured types are built recursively on primitive types $\mathsf{P} \in \mathcal{P}$ and/or structured types:

- Product types: $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$
- Sets: $\mathsf{Set}(\mathsf{T})$
- Multisets: $\mathsf{MSet}(\mathsf{T})$
- Lists: $\mathsf{List}(\mathsf{T})$
- Mappings: $\mathsf{S} \Rightarrow \mathsf{T}$

Here $\mathsf{S}, \mathsf{T}, \mathsf{T}_1 \ldots \mathsf{T}_n \in \mathcal{T}$ denote any type in $\mathcal{T}$. The representation of any data structure reconstructed in types of this type system $\mathcal{T}$ is a model of such data structure. If the difference calculus work on this new data type successfully, this data type will become a new structured data type in this type system, then it can be used by other data structures. Hence the type system can be extended by adding more structured data types.

b) Differences and Difference Types

Difference Types are a mathematical model of differences for the purpose of implementing effective version control for artifacts. For the moment we leave the structure of the underlying types aside. We consider objects of any type T in the type system $\mathcal{T}$. To express that u is an artifact of type T, we write for short $u\colon \mathsf{T}$. Let $u, v\colon \mathsf{T}$ be intermediate versions created in building an artifact.

A typed directed difference operator δ_T has been defined to generate the difference between two artifacts of type T, and Δ_T denotes the type of this difference.

$$\delta_\mathsf{T}\colon \mathsf{T} \times \mathsf{T} \to \Delta_\mathsf{T}$$

That means applying δ_T to $u, v\colon \mathsf{T}$, yields their directed difference, which is of type Δ_T:

$$u, v\colon \mathsf{T} \vdash \delta_\mathsf{T}(u, v)\colon \Delta_\mathsf{T} \tag{1}$$

For many structured data types, multiple possible interpretations exist for the actual difference in values. Therefore an algorithmic solution requires a selection criteria. We have choosen the approach of "minimal difference" or "greatest possible similarity" to be such criteria. This leads to the introduction of a similarity measure.

c) Similarity Measure

$\mathbb{M} = [0, 1]$, a closed interval of real number, denotes the type of similarity measure between two artifacts, 0 indicates they are completely different and 1 indicates they are equal. μ_T is the typed measurement operator. By applying the measurement operator to these two artifact of type T, a value of type $\mathbb{M}$ is generated:

$$\mu_\mathsf{T}\colon \mathsf{T} \times \mathsf{T} \to \mathbb{M}$$

The similarity measure between two artifacts, $u, v\colon \mathsf{T}$, which have been compared completely is generated by applying μ_T to them:

$$u, v\colon \mathsf{T} \vdash \mu_\mathsf{T}(u, v)\colon \mathbb{M} \tag{2}$$

However algorithmically it is often sufficient to compare objects only to a certain degree. When two objects over type T have not been completely compared, the partial similarity measure, which is denoted by $\mathcal{M}$, can be interpreted as the type of subintervals of [0,1], is generated by applying an estimated similarity measurement operator $\widetilde{\mu}_\mathsf{T}$:

$$\widetilde{\mu}_\mathsf{T}\colon \mathsf{T} \times \mathsf{T} \to \mathcal{M}$$

If $u, v\colon \mathsf{T}$ have not been completely compared, by applying $\widetilde{\mu}_\mathsf{T}$ to u and v, the partial similarity measure is generated:

$$u, v\colon \mathsf{T} \vdash \widetilde{\mu}_\mathsf{T}(u, v)\colon \mathcal{M} \tag{3}$$

We can always assume that $\mu_\mathsf{T}(u, v) \in \widetilde{\mu}_\mathsf{T}(u, v)$. One could describe $\widetilde{\mu}_\mathsf{T}(u, v) = [\underline{\mu}_\mathsf{T}(u, v), \overline{\mu}_\mathsf{T}(u, v)]$ in which $\underline{\mu}_\mathsf{T}(u, v)$ denotes lower bound of $\mu_\mathsf{T}(u, v)$ and $\overline{\mu}_\mathsf{T}(u, v)$ denotes upper bound of $\mu_\mathsf{T}(u, v)$[1].

d) Difference Calculus

Clearly, if the directed difference between two versions of an artifact can be determined, it would be useful to find a way of generating one version by applying this difference to the other version. To do this an additional operator, the application operator, $\odot_\mathsf{T}$, needs to be defined:

$$\odot_\mathsf{T}\colon \mathsf{T} \times \Delta_\mathsf{T} \to \mathsf{T}$$

If for some $u, v\colon \mathsf{T}$, δ_T(u,v) can be determined, then using $\odot_\mathsf{T}$ this difference can be applied to u to generate v:

$$u, v\colon \mathsf{T} \vdash v = u \odot_\mathsf{T} \delta_\mathsf{T}(u, v) \tag{4}$$

A system of $\{\mathsf{T}, \Delta_\mathsf{T}, \mathbb{M}, \mathcal{M}, \delta_\mathsf{T}, \mu_\mathsf{T}, \widetilde{\mu}_\mathsf{T}, \odot_\mathsf{T}\}_{\mathsf{T} \in \mathcal{T}}$ is called a difference calculus when it fulfills the above (1), (2), (3) and (4).

[1]There are many ways of describing the difference between two artifacts. Each will result in a particular similarity. We define the difference for structured types so as to maximise the apparent similarity.

III Similarity Measure

In this section, the initial similarity measure is defined across all types in the type system $\mathcal{T}$. The similarity measure of a structured data type is determined from the similarity measure of types underlying it.

a) Primitive Types

Primitive types $\mathsf{P} \in \mathcal{P}$ have a decidable equality. For the purpose of this study, primitive types are viewed as having an atomic behaviour. The similarity measure of objects of primitive types is either 1 or 0, depending on the object being equal or not.

Definition III.1

Let x, y: P be objects of type P. By applying the typed similarity measurement operator μ_{P} to them, the similarity measure between x, y is defined as:

$$\mu_{\mathsf{P}}(x,y) =_{\mathsf{def}} \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

b) Product Types

A product type containing n elements is denoted by $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$. The similarity measure of objects of type $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ is determined by the average of the similarity measures of the corresponding components[2].

Definition III.2

Let A, B: $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ be two objects of n-tuple product type with $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$. The similarity measure between A and B is defined by the average of the similarity measures between correponding components:

$$\mu_{(\mathsf{T_1} \times \ldots \times \mathsf{T_n})}(A,B) =_{\mathsf{def}} \frac{1}{n}(\mu_{\mathsf{T}_1}(a_1,b_1) + \ldots + \mu_{\mathsf{T}_n}(a_n,b_n)) = \frac{1}{n}\sum_{i=1}^{n} \mu_{\mathsf{T}_i}(a_i,b_i)$$

This can be extended to intervals by means of standard interval arithmetic.

c) Sets

$\mathsf{Set}(\mathsf{T})$ denotes finite typed sets whose elements are of type T. Let A: $\mathsf{Set}(\mathsf{T})$ be such a set, $|A|$ denotes the size of set A. Such a set structure has a boolean membership (also called characteristic function).

[2]One could argue that a more realistic similarity measure would be the weighted average of similarities where the weight is related to the "size" of objects. This will be further investigated once experimental data are available.

To define the similarity measure of two sets in general, all possible mappings between elements have to be considered and the one with the maximum similarity measure would be choosen.

To compare the similarity measure of two sets A and B, four cases need to be considered. The first case is when A and B both are empty sets, the similarity between them is 1. The second case is when A is an empty set while B is a non-empty set, the similarity measure is 0. The third case is when A is a non-empty set while B is an empty set, the similarity measure is also 0. The last case is when A and B both are non-empty sets. In order to identify the similarity measure between them, maximal non-empty subsets $X \subseteq A$ and $Y \subseteq B$ with $|X| = |Y| = \min(|A|, |B|)$ and one-to-one mappings $\varphi : X \leftrightarrow Y$ are considered.

In the last case, if A and B are same size sets, $X = A$ and $Y = B$, otherwise, the size of X or Y is the smaller size set between A and B. In this way, most similar elements pairs between A and B are identified. Therefore the similarity measure between two sets is defined below:

Definition III.3

Let A, B: $\mathsf{Set}(\mathsf{T})$ be sets whose elements are of type T, when A and B both are non-empty sets, non-empty subset X and Y, $X \subseteq A$, $Y \subseteq B$, $|X| = |Y| = \min(|A|, |B|)$ and one-to-one mapping $\varphi : X \leftrightarrow Y$ are considered.

The similarity measure, σ_{max}, between non-empty sets A and B is defined as the maximum value of the average of the similarity measure between subsets X and Y:

$$\sigma_{\mathsf{max}}(A,B) =_{\mathsf{def}} \max_{\substack{\varphi: X \leftrightarrow Y \\ X \subseteq A, Y \subseteq B \\ |X| = |Y|}} \left(\frac{\sum_{x \in X} 2\mu_{\mathsf{T}}(x, \varphi(x))}{|A| + |B|} \right)$$

Therefore the similarity measure between two sets is defined:

$$\mu_{\mathsf{Set}(\mathsf{T})}(A,B) =_{\mathsf{def}} \begin{cases} 1 & \text{if } A = \emptyset \text{ and } B = \emptyset \\ 0 & \text{if } A = \emptyset \text{ and } B \neq \emptyset \\ 0 & \text{if } A \neq \emptyset \text{ and } B = \emptyset \\ \sigma_{\mathsf{max}}(A,B) & \text{if } A \neq \emptyset \text{ and } B \neq \emptyset \end{cases}$$

Again the similarity measure between two sets can be extended to the partial similarity measure using interval arithmetic.

IV Directed Difference

In section III, the similarity measure of two objects of the same data type has been reduced to the similarity measure of their respective components. The *Directed Difference* between objects of each

data type is discussed in this section.

The directed difference of any type T, whose type is denoted as Δ_{T}, is derived by comparing two artifacts of type T. When comparing two identical objcets, the original one is returned. When we do not get a handle on the structure of the type T, and we are comparing two different objects, the directed difference can be described as a pair consisting of these objects, or similarly a structure built on the directed differences of components of these objects. Hence the type of directed difference has a structure like:

$$\Delta_{\mathsf{T}} \approx \mathsf{T} \mid \mathsf{T} \times \mathsf{T}$$

When we know something about the structure of a type, i.e. $\mathsf{T} = \tau(\mathsf{S})$, the type of directed difference takes the shape:

$$\Delta_{\tau(\mathsf{S})} \approx \tau(\mathsf{S}) \mid \tau'(\Delta_{\mathsf{S}})$$

a) Primitive Types

For this paper, any type we do not want to investigate further for the purpose of determining the difference can be considered primitive.

The type of the directed difference of a primitive type P, is defined as:

$$\Delta_{\mathsf{P}} =_{\mathsf{def}} \mathsf{P} \mid \mathsf{P} \times \mathsf{P}$$

Definition IV.1

Let x, y: P be objects of primitive type $\mathsf{P} \in \mathcal{P}$, the directed difference between x and y depends on whether or not they are identical. If $x = y$, then the directed difference is x which refers that x is not changed, otherwise it is defined as (x, y) which indicates that x is replaced by y.

$$\delta_{\mathsf{P}}(x, y) =_{\mathsf{def}} \begin{cases} x & \text{if } x = y \\ (x, y) & \text{otherwise} \end{cases}$$

b) Product Types

Product types $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ $(n \geqslant 2)$ are the simplest structured data type. The directed difference between instances of n-tuple product type can be determined by applying the corresponding typed directed difference operator to each element of them separately. Let A, B: $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ be objects of n-tuple of product type $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ with $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$. The type of the directed difference between them is a n-tuple of directed differences:

$$\Delta_{\mathsf{T_1} \times \ldots \times \mathsf{T_n}} =_{\mathsf{def}} (\mathsf{T_1} \times \ldots \times \mathsf{T_n}) \mid \Delta_{\mathsf{T_1}} \times \ldots \times \Delta_{\mathsf{T_n}}$$

The directed difference between A and B can be determined by applying the typed directed difference operator of n-tuple product type, $\delta_{\mathsf{T_1} \times \ldots \times \mathsf{T_n}}$, to A and B.

Definition IV.2

Let A, B: $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ be objects of n-tuple of product type $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ with $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, the directed difference between A and B is defined as:

$$\delta_{\mathsf{T_1} \times \ldots \times \mathsf{T_n}}(A, B) =_{\mathsf{def}} \begin{cases} A & \text{if } A = B \\ (\delta_{\mathsf{T_1}}(a_1, b_1), \ldots, \delta_{\mathsf{T_n}}(a_n, b_n)) & \text{otherwise} \end{cases}$$

The equality of two objects over type $\mathsf{T_1} \times \ldots \times \mathsf{T_n}$ is determined by the identity of each corresponding component. By comparing each corresponding elements in these two n-tuples, if the componentwise directed difference has been generated as $\delta_{\mathsf{T_1}}(a_1, b_1) = a_1, \ldots, \delta_{\mathsf{T_n}}(a_n, b_n) = a_n$, then $\delta_{\mathsf{T_1} \times \ldots \times \mathsf{T_n}}(A, B) = (a_1, \ldots, a_n) = A$, hence $A = B$.

c) Sets

Sets are the fundamental data structure in mathematics. From one set to the other set, the mathematical concept of the difference between two sets is already well known, in fact the operation $A \backslash B$ is defined as a set which contain the elements of A which are not in B. This already well established operation proves to be useful in creating our directed difference methodology.

The simplest interpretation of the directed difference between A and B is as a triple consists of the set of common elements (i.e. $A \cap B$), the set of elements to be deleted from A (i.e. $A \backslash B$) and the set of elements that are used to be inserted (i.e. $B \backslash A$). For sets over primitive types this classical set theoretical approach suffices. However if an element of a set has been changed, do we consider this as a deletion of an element and the insertion of a new one? What if this was only a minor change? Here one has to decide, and we choose:

$$\underbrace{\mathsf{Set}(\Delta_{\mathsf{T}})}_{\text{changed}} \times \underbrace{\mathsf{Set}(\mathsf{T})}_{\text{deleted}} \times \underbrace{\mathsf{Set}(\mathsf{T})}_{\text{inserted}}$$

In this triple, the first set is a set of the directed difference between single elements in A and B, it contains the common elements and the difference elements with only small changes, the second set is composed of all deleted elements in A, the third set contains all newly inserted elements.

Then the directed difference of finite typed sets depends on whether or not they are identical. If two sets have been determined as two identical ones, the directed difference will return the original set as an identical indicator. Otherwise it is defined as a triple, which includes a single element directed difference set, a deleted set and a inserted set. Hence the type of the directed difference between two finite typed sets of type $\mathsf{Set}(\mathsf{T})$ is defined

below:

$$\Delta_{\mathsf{Set}(\mathsf{T})} =_{\mathsf{def}} \mathsf{Set}(\mathsf{T}) \mid \mathsf{Set}(\Delta_{\mathsf{T}}) \times \mathsf{Set}(\mathsf{T}) \times \mathsf{Set}(\mathsf{T})$$

The similarity measure between two sets has been identified by finding the most similar pairs between these two sets. The most similar pair has the smallest difference, hence the first set in the directed difference is composed of the directed difference between pair of most similar elements.

However, when defining the directed difference between two sets, whether or not they are empty still needs to be considered.

Definition IV.3

Let A, B: $\mathsf{Set}(\mathsf{T})$ be finite typed sets, the directed difference depends on whether or not they are identical.

When A and B both are non-empty sets, let $X_{\mathsf{max}}, Y_{\mathsf{max}}$: $\mathsf{Set}(\mathsf{T})$ be maximal non-empty subsets, i.e. $X_{\mathsf{max}} \subseteq A$, $Y_{\mathsf{max}} \subseteq B$ with $|X_{\mathsf{max}}| = |Y_{\mathsf{max}}| = \min(|A|, |B|)$ and $\varphi_{\mathsf{max}} : X_{\mathsf{max}} \leftrightarrow Y_{\mathsf{max}}$ one-to-one mapping, such that:

$$\frac{\sum_{x \in X_{\mathsf{max}}} 2\mu_{\mathsf{T}}(x, \varphi_{\mathsf{max}}(x))}{|A| + |B|} = \sigma_{\mathsf{max}}(A, B)$$

Let D_{max}: $\mathsf{Set}(\Delta_{\mathsf{T}})$ be the set of directed differences between single element in X_{max} and Y_{max} with $D_{\mathsf{max}} = \{\delta_{\mathsf{T}}(x, \varphi_{\mathsf{max}}(x)) \mid x \in X_{\mathsf{max}}\}$.

Then the directed difference, $\delta_{\mathsf{max}}(A, B)$ is defined as:

$$\delta_{\mathsf{max}}(A, B) =_{\mathsf{def}} (D_{\mathsf{max}}, A \backslash X_{\mathsf{max}}, B \backslash Y_{\mathsf{max}})$$

If $A = B$, then $X_{\mathsf{max}} = Y_{\mathsf{max}} = A = B$, φ_{max} is the identity, and hence $\delta_{\mathsf{max}}(A, A) = (A, \emptyset, \emptyset)$.

Therefore the directed difference between two sets is defined:

$$\mu_{\mathsf{Set}(\mathsf{T})}(A, B) =_{\mathsf{def}} \begin{cases} \emptyset & \text{if } A = \emptyset \text{ and } B = \emptyset \\ (\emptyset, \emptyset, B) & \text{if } A = \emptyset \text{ and } B \neq \emptyset \\ (\emptyset, A, \emptyset) & \text{if } A \neq \emptyset \text{ and } B = \emptyset \\ \delta_{\mathsf{max}}(A, B) & \text{if } A \neq \emptyset \text{ and } B \neq \emptyset \end{cases}$$

V Application

The Difference Calculus is a theoretical framework which has been implemented by a Java program using Java generics. All types and operators of each type have been developed in this Java application. In order to apply this difference calculus in a domain, a mathematical model of this domain has to be created in Java based on types in the type system $\mathcal{T}$. By doing this the equality, the similarity measure and the directed difference between instances of this model can be computed simultaneously. We demonstrate this in the following with a simple graph example.

A finite typed graph has been modelled as a set of pairs, where the first element in each pair is a vertice, the second element in each pair is a set of nodes which are linked to that vertice.

$$\mathsf{Graph}(\mathsf{T}) =_{\mathsf{def}} \mathsf{Set}(\mathsf{T} \times \mathsf{Set}(\mathsf{T}))$$

This graph model has been built in the Java application. In order to explain the process, an example of comparing two graphs is illustrated below.

Let A, B: $\mathsf{Graph}(\mathbb{S})$ be graphs[3] (see Figure 1). When comparing A and B, the one-to-one mapping with the maximum similarity measure needs to be determined. To achieve this, all possible mappings between elements in them are identified, then the most similar pairs are chosen.

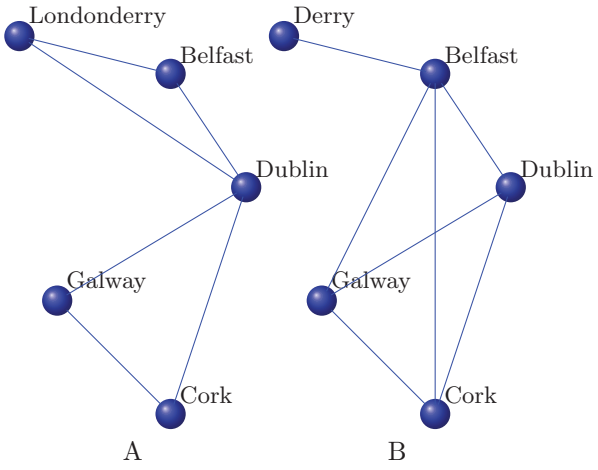

Fig. 1: Models of Two Simple Graphs of Travel Maps.

$A = \{$(`Dublin`, {`Belfast`,`Galway`,`Cork`,`Londonderry`}),
(`Cork`, {`Dublin`,`Galway`}),
(`Galway`, {`Dublin`,`Cork`}),
(`Belfast`, {`Londonderry`,`Dublin`}),
(`Londonderry`, {`Dublin`,`Belfast`}),
$\}$

$B = \{$(`Belfast`, {`Dublin`,`Galway`,`Cork`,`Derry`}),
(`Derry`, {`Belfast`}),
(`Dublin`, {`Belfast`,`Galway`,`Cork`}),
(`Galway`, {`Dublin`,`Belfast`,`Cork`}),
(`Cork`, {`Dublin`,`Belfast`,`Galway`}),
$\}$

Each element in A and B is of type $\mathbb{S} \times \mathsf{Set}(\mathbb{S})$, hence the similarity measures between every two

[3] $\mathsf{Graph}(\mathbb{S})$ is then a model of a graph with vertices labeled with strings s from the primitive type $\mathbb{S}$.

elements in A and B are calculated in terms of the definition III.2 and III.3. Therefore, the one-to-one mapping with the maximum simialrity measure is determined:

A	B	Similarity
Dublin	Dublin	$\frac{1}{2} * (1 + \frac{2*3}{4+3}) \simeq 0.9286$
Galway	Galway	$\frac{1}{2} * (1 + \frac{2*2}{2+3}) = 0.9$
Cork	Cork	$\frac{1}{2} * (1 + \frac{2*2}{2+3}) = 0.9$
Belfase	Belfast	$\frac{1}{2} * (1 + \frac{2*1}{2+4}) \simeq 0.6667$
Londonderry	Derry	$\frac{1}{2} * (0 + \frac{2*1}{2+1}) \simeq 0.3333$

Then in terms of definition III.3, the similarity measure between A and B is 0.7457.

$$\frac{2}{10} * (0.9286 + 0.9 + 0.9 + 0.6667 + 0.3333) \simeq 0.7457$$

This similarity result shows that two graphs are not identical. At the same time, the directed difference is identified to indicate the matching nodes, the deletion and the insertion of routes (see Figure 2).

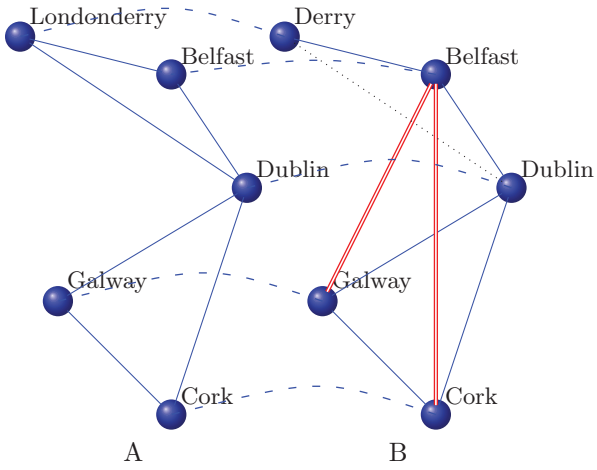

Fig. 2: Results of Comparing Two Graphs Models.

Applying the difference calculus to the refactoring of graphs indicates a particular association between notes in two non-isomorphic graphs.

VI Conclusions

The results of applications indicate that it is possible to use a universal approach to develop a version control tool for arbitrary structured data. By applying the Design Calculus to objects modelled on the type system, the directed difference and the similarity measure are obtained. Work continues to complete the framework and to apply this Design Calculus to more complex structured data.

References

[1] M. J. Rochkind, "The source code control system," *IEEE Transactions on Software Engineering*, vol. SE-1, no. 4, pp. 364–370, December 1975.

[2] W. F. Tichy, "Design, implementation, and evaluation of a revision control system," *Proceedings of the 6th international conference on Software engineering*, pp. 58–67, 1982.

[3] S. Dart, "Concepts in configuration management systems," *Proceeding of the 3rd international workshop on Software configuration management, SCM 1991*, pp. 1–18, 1991.

[4] J. Estublier, D. Leblang, G. Clemm, R. Conradi, W. Tichy, A. Hoek, and D. Wiborg-Weber, "Impact of software engineering research on the practice of software configuration management," *ACM Transactions on Software Engineering and Methodology*, vol. 14, pp. 1–48, October 2005.

[5] W. F. Tichy, "Rcs a system for version control," *Purdue University, Purdue e Pubs*, vol. 15, no. 7, pp. 637–654, March 1984.

[6] G. Cobena, S. Abiteboul, and A. Marian, "Detecting changes in xml documents," in *Data Engineering, 2002. Proceedings. 18th International Conference on.* IEEE, 2002, pp. 41–52.

[7] J. Hunt, "An approach for the automated risk assessment of structural differences between spreadsheets (diffxl)," *Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG) 2009*, no. ISBN: 978-1-905617-89-0, 2009.

[8] C. Chambers, M. Erwig, and M. Luckey, "Sheetdiff:a tool for identifying changes in spreadsheets," *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 85–92, November 2010.

[9] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change detection in hierarchically structured information," *Proceedings of the 1996 ACM SIGMOD international conference on Management of data, SOGMOD 1996*, vol. 25, no. 2, pp. 493–504, June 1996.

[10] S. S. Chawathe and H. Garcia-Molina, "Meaningful change detection in structured data," *Proceedings of the 1997 ACM SIGMOD international conference on Management of data, SOGMOD 1997*, vol. 26, no. 2, pp. 26–37, June 1997.

[11] E. Lempsink, S. Leather, and A. Loh, "Type-safe diff for families of datatypes," in *Proceedings of the 2009 ACM SIGPLAN workshop on Generic programming*, 2009, pp. 61–72.