WiClean: A System for Fixing Wikipedia Interlinks Using Revision History Patterns

Stephan Goldberg[1] Tova Milo[1] Slava Novgorodov[2] Kathy Razmadze[1]
[1]*Tel Aviv University* [2]*eBay Research*
{stephank, kathyr}@mail.tau.ac.il milo@post.tau.ac.il snovgorodov@ebay.com

ABSTRACT

We present WICLEAN, a Wikipedia plug-in that supports the identification and cleaning of certain types of errors in Wikipedia interlinks. The system mines update patterns in Wikipedia revision logs, identifies the common time frames in which they occur, and employs them to signal incomplete/inconsistent updates and suggests corrections. We demonstrate the effectiveness of WICLEAN in identifying actual errors in a variety of Wikipedia entity types, interactively employing the VLDB'19 audience as editors to correct the identified errors.

PVLDB Reference Format:
Stephan Goldberg, Tova Milo, Slava Novgorodov, Kathy Razmadze. WiClean: A System for Fixing Wikipedia Interlinks Using Revision History Patterns. *PVLDB*, 12(12): 1846-1849, 2019.
DOI: https://doi.org/10.14778/3352063.3352081

1. INTRODUCTION

Wikipedia, the free-content web encyclopedia, is one of the most popular web-sites on the Internet. According to Time magazine, Wikipedia's "open-door policy" of allowing anyone to edit the data, has made it the largest and possibly the best encyclopedia in the world. At the same time, the continuously evolving content, constantly updated by a large number of uncoordinated users, makes the maintenance of a clean, consistent encyclopedia an extremely challenging task. To understand the volume of updates, the English Wikipedia, as of the end of 2017, consists of 5.5 million articles, with an average of 3.4 million edits monthly, by around 31,000 users. The goal of the WICLEAN system presented here is to assist Wikipedia editors in this challenging task. Specifically, we focus in this work on the correctness of Wikipedia *inter-links* that point from one article to another. Such inter-links form a major component of the structured part of Wikipedia (in particular infoboxes and tables) and their correctness is critical for coherent browsing.

The maintenance of correct and consistent links is challenging for two main reasons. First, as different Wikipedia pages are often edited by different people, it could be that not all the updates performed in one page are properly propagated to other related pages. For example, consider the Wikipedia page of the soccer player Neymar. The links in its infobox point to the page of his current team - Paris Saint Germain (PSG), to his place of birth, and so on. Similarly, the *squad* table at the PSG Wikipedia page points back to Neymar's. When Neymar moved to PSG in the summer of 2017, leaving his previous team Barcelona, each of the three related pages (Neymar's, PSG, Barcelona) had to be updated to keep the Wikipedia up to date and consistent. Indeed, even when considering a single page, many of the small edits done by individual users are conceptually a part of a larger update which, if not fully executed, leaves the article outdated and/or inconsistent. Continuing with our example, not only the *current club* link in Neymar's page should be updated but also the links list in his *career* table (to add his current position), and possibly also his residence link.

A second challenge is that the consistency constraints on Wikipedia inter-links are often soft, and need not be applied at all times. To continue with our example, soccer players often switch teams in the summer and winter, in periods referred to as *transfer windows*. Transfers take long time to be officially confirmed, with typically many rumors in the background. Consequently, in the time period between the beginning of the transfer window and the official approval of the transfer, there may be hundreds of Wikipedia edits to the players Wikipedia pages, adding/removing new/old links to the expected teams, reverting previous edits and making new ones. It is often only after the transfer is officially approved that the corresponding Wikipedia team articles (of the old and new teams) are also updated. The Wikipedia articles thus appear to be inconsistent within the transfer window, *but this in fact is considered a positive phenomenon* as it allows readers to view the most up to date information at the given point in time. Figure 1 illustrates a set of edits taking place within such a transfer window. We see here a portion of the revision history of several Wikipedia articles (of soccer players and teams), merged together into one timeline. The Subject column identifies the article where the addition/removal of a link occurred, the Object column identifies the article to which the added/deleted links point to, and the Relation column describes the link type. We can see here that, for Neymar, after several edits and reverts, the transfer is reflected in both his and the team's pages.

The thesis underlying WICLEAN is that Wikipedia updates often follow desirable patterns and lead to consistent states. WICLEAN thus mines Wikipedia revision logs to identify common update patterns, along with time windows in which they typically occur. Potential errors are then identified by signaling updates that deviate from the mined patterns.

Formally, we model Wikipedia entities (articles) and the links between them as a graph. Nodes and edges are labeled by type names. Intuitively, the revision history of each article records the edits made to the outgoing links of the corresponding graph node. Given an entity type of interest, our algorithm identifies meaning-

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352081

#	+/-	Subject	Relation	Object	Time
1	-	Neymar	current_club	Barcelona	...1531
2	-	Neymar	league	La Liga	...8711
3	-	Barcelona	squad	Neymar	...2804
4	+	Neymar	current_club	PSG	...3321
5	+	PSG	squad	Neymar	...8263
6	+	Barcelona	squad	Neymar	...4040
7	+	Neymar	league	Ligue 1	...8711
8	+	Neymar	current_club	Barcelona	...5861
9	-	Kylian Mbappe	current_club	Monaco	...9459
10	-	Neymar	current_club	PSG	...3732
11	-	Neymar	current_club	Barcelona	...6109
12	+	Neymar	current_club	PSG	...7694
13	-	Barcelona	squad	Neymar	...8001
14	+	Kylian Mbappe	current_club	PSG	...9589

Figure 1: Actions from revision history of several articles

ful relevant edit patterns across revision histories of entities of the same or other types, along with time windows in which partial edits are tolerable. An iterative refinement process, of both patterns and windows, allows WICLEAN to efficiently focus on relevant related entity types (and their revision histories) and time frames. The discovered windows and patterns are then used by WICLEAN to assist Wikipedia editors in correcting/updating Wikipedia links. The WICLEAN system both alerts Wikipedia editors on past edits that appear to be incomplete as well as provides users with on-line assistance as they update the encyclopedia.

Demonstration Overview. We will demonstrate the operation of WICLEAN on a real-life snapshot of Wikipedia's revision history, and actual errors identified by the system, employing the VLDB'19 attendees as editors for their correction. We will demonstrate the mining of update patterns (and their time windows) in a variety of domains including sports and cinematography, then show how incomplete/inconsistent updates are identified and corrected.

2. TECHNICAL BACKGROUND

We briefly present the data model and the algorithms underlying the WICLEAN system.

2.1 Preliminaries

Wikipedia Graph. We model the relationships between Wikipedia entities at a given point in time using a graph $G(V, E)$. Each node in V represents a Wikipedia entity and is labeled by a unique name (e.g. Neymar) and an entity type (e.g. *soccer_player*). Each edge in E represents a relationships between two entities and is labeled by the relationship name (e.g. $current\_club$).

We use an alignment from Wikipedia entities to DBPedia [1] to derive the entity types. In general, the types belong to type taxonomy - the higher the type is in the taxonomy the more general it is - and an entity may have multiple types. For two types t, t' we use $t' \leq t$ to denote the fact that t either equals to t' or generalizes it. For example, $soccer\_player \leq athlete \leq person$. We assume that each entity e has one most specific type to which it belongs[1] and use it as its label, denoted $type(e)$. For a type t we use $entities(t)$ to refer to all entities labeled by a type $t' \leq t$.

Actions and Inverse actions. The revision history of Wikipedia entities contains edits to the graph edges. We particularly consider two types of actions: adding *new* edges and deleting *existing* ones. Our model associates each action with a time stamp. We use a triplet of the form $a = (+, (u, l, v), t)$ (resp. $a = (-, (u, l, v), t)$) to denote the addition (rep. deletion) of edge from u to v with label l at time t.

We say that an action a' is the *inverse* of a preceding action a, denoted $a' = Inv(a)$ if applying a' after a leaves the graph unchanged. For instance, in Figure 1, action #6 is an inverse action of action #3.

(Reduced) set of actions. Given a Wikipedia graph $G(V, E)$, a set of entities $S \subseteq V$, and a time frame (called a *window*), we consider the set of all actions (denoted as A) that were recorded in the revision history of the entities in S, within the given window.

In the update processes some edits may naturally be reversed. To consider only the final effect we focus on *reduced actions sets* that do not include an action and its inverse. More formally, given a graph G, we say that two actions sets are *equivalent* if, when the actions are applied on G in the order of their timestamps, they generate the same result graph. Given an action set it is easy to derive an equivalent reduced variant by iteratively removing actions and their inverse. To continue our example, the action set in Figure 1 is equivalent to the set of gray actions (lines) on the same figure.

Note that up to possibly different time stamps, the reduced version obtained through this iterative removal process is unique, namely contains the same set of graph update operations. Furthermore, the time stamps are no longer important as any permutation of the actions yields the same output graph. We thus consider from now on only reduced sets of actions and ignore the time stamps of the actions in the set, referring to actions as pairs $a = (op, (u, l, v))$ where $op \in \{+, -\}$.

Abstract Actions and Patterns. Since we are trying to find general update patterns across our Wikipedia graph, we want to generalize a set of actions involving specific entities to general patterns over the corresponding entity types. For that we define the notion of *abstract actions*. We associate with each entity type t an infinite set of variables $t_1, t_2, \ldots$. Then, an *abstract action* is the pair of the form $a = (op, (t', l, t''))$ where $op \in \{+, -\}$ and t', t'' are type variables and l is an edge label, and a *pattern* is a set of abstract actions. Finally, given a pattern p we say that a set A' of (concrete) actions is a *realization* of p if A' may be obtained from p by replacing each variable of type t by some graph node in $entities(t)$. To illustrate that, lines number 11,12,13,5,2,7 in Figure 1 are the realization of the pattern shown in Figure 2.

Given an entity type t, we are interested in entities' updates that are related (possibly transitively) to entities of type t. We thus focus on *connected* patterns, where the updated edges are related. Formally,

Definition 2.1. *Given an update pattern p, let g_p be the directed graph whose nodes are the type variables in p and where an edge from node t_1, t_2 exists in g_p iff p includes an abstract action of the form $(op, (t_1, l, t_2))$. Given a type t, we say that a pattern p is* ***connected*** *(w.r.t t) iff the graph g_p is connected, and each node in it is reachable from some node of type t.*

For example, the pattern shown in Figure 2 is connected. But if we replace the variable $player_1$ in lines 11 and 13 by a new variable $player_2$, then it becomes disconnected (and composed of two smaller connected patterns - the abstract actions in lines $10, 5, 2, 7$ and the abstract actions in lines $11, 13$).

As we are only interested in connected patterns, for brevity, unless stated otherwise, whenever we use below the term pattern we mean a *connected* one.

Frequent Patterns. Given a type t and a set A of actions, the *frequency* of a pattern p (w.r.t to t and A), is the fraction of entities of type t that participate in realizations of p in A. Formally,

[1]Otherwise we can add to the taxonomy a new type name which represents the intersection of the multiple most specific types of the given entity, which will play the role of this most specific type.

#	Edit type	Subject	Relation	Object
10	-	$player_1$	current_club	$team_1$
11	+	$player_1$	current_club	$team_2$
5	-	$team_1$	squad	$player_1$
13	+	$team_2$	squad	$player_1$
2	-	$player_1$	league	$league_1$
7	+	$player_1$	league	$league_2$

Figure 2: Pattern found from set of action in Figure 1

Definition 2.2. *The* frequency *of a pattern p in a set of actions A, w.r.t to a an entity type t, is defined as* $frequency(A,t,p) = \frac{|\{e \in entities(t) | e \text{ appears in } A' \subseteq A, A' \text{ is a realization of } p\}|}{|entities(t)|}$

To continue with our running example, consider the actions in Figure 1 and the pattern in Figure 2, and assume there are overall five players in Wikipedia. The frequency of this pattern in the given actions set is 0.2 because there is only one player (Neymar) that the patterns holds for, out of the five existing players. However the frequency of the partial pattern displayed in figure 2 in lines 1 and 2 (gray lines), in this actions set is 0.4 because there are 2 players for which that the patterns holds.

Given a type t, a set A of actions and frequency threshold τ we will be interested in finding patterns whose frequency in A is above the threshold. To avoid redundancy, we would like to consider only the most *specific* such patterns. Formally, we say that a pattern p is more specific than a pattern p' (alternatively, p' is more general than p), denoted $p \prec p'$, if p' may be obtained from p by removing some abstract actions, replacing some type variables in p by variables of a more general type, or both. To illustrate, for the patterns

$p_1 = \{(-, (player_1, current\_club, team_1)),$
$(+, (player_1, current\_club, team_2))\}$
$p_2 = \{(-, (athlete_1, current\_club, team_1)),$
$(+, (athlete_1, current\_club, team_2))\}$
$p_3 = \{(-, (athlete_1, current\_club, team_1))\}$

we have that $p_1 \prec p_2 \prec p_3$.

Thus, given a type t and a set A of actions our goal will be to find the most specific patterns with frequency above a given threshold.

2.2 Finding Windows and Frequent Patterns

Intuitively, given an entity type t of interest, we wish to signal out significant time frames and identify the most specific frequent patterns in them. Frequent itemset mining is known to be a computationally difficult task [11] and the same hardness bounds hold for our problem (The proof works by reduction from the general itemset mining problem). Yet, we manage to derive a scalable, highly parallelizable solution due to the following implementation choices.

First, recall that the revision histories are distributed across the entities and their overall size can be very large. However, as we shall see below, our focus on *connected* patterns allows to consider only those entity types (and their corresponding revision histories) that may potentially be related to the input type t via frequent edit patterns, thereby significantly pruning the search space. Moreover, different types may be processed in parallel. Second, we restrict our attention to non-overlapping time windows, and split the revision histories accordingly. This reduces the number of actions to be considered for each window and allows to parallelize the processing of the action sets in the different windows. Our experiments with real Wikipedia data indicates that this is a reasonable design choice as in practice for an input type t there are very few meaningful (update-wise) time frames that overlap, and those can be merged into a somewhat longer window that includes both update patterns.

Given a type t (e.g. soccer players), a minimal time unit W_{min} (e.g. one day), and a frequency threshold τ (e.g 50%), our algorithm first focuses on windows of size W_{min}, considering only edit logs of entities of type t. The identified patterns (or lack of) are then used to iteratively extend the considered entity types, windows, and patterns. The algorithm is intuitively sketched below.

Initialization. As above, given an entity type, our initial entity set S contains all entities of the given type. Users not familiar with the type hierarchy may provide a seed entity e and the system will use $type(e)$ as an input. As mentioned, to derive $type(e)$ we use an alignment from Wikipedia entities to DBPedia [1]. Then to find all entities of type t we employ a corresponding inverse index.

We first split the timeline into consecutive time frames of size W_{min}, considering for each such window w the set of edit actions $A_{S,w}$ performed on the entities in S within the window w. For efficiency we first restrict out attention to singleton patterns (consisting of a single abstract action), and identify those windows with patterns above the threshold frequency. As noted above, the processing of the individual windows is independent and may be parallelized. In the absence of qualifying windows, depending on the user preferences we extend W_{min} and/or reduce τ, and repeat. Once some qualifying windows are identified, we proceed iteratively as follows.

Refinement. We alternate between extending the considered entity types and refining the considered windows/frequency threshold until optimized result is achieved.

Let w be a window identified in the previous iteration with a corresponding set P of frequent patterns (The processing of different windows is performed in parallel). We first extend the entity set S to include all entities of the types occurring in P (if not already in the set), and add their revision history within w to the set of considered actions. We then mine the extended set of actions $A_{S,w}$ for frequent patterns (we explain below how this is done). The extension is repeated as long as the new identified patterns include new entity types. Next, to optimize the window size and the frequent threshold, we examine the effect of extending (resp. lowering) the window boundaries (frequency threshold). The extension granularity (resp. frequency bound reduction) may be determined by the user. We repeat the above two steps if further patterns are discovered, and otherwise return the most specific identified patterns.

Patterns Mining. We employ here an iterative incremental algorithm. Intuitively, starting from singleton (most specific) frequent actions, we consider for each (previously discovered) pattern p its graph g_p from definition 2.1 and attempt to extend it with a new edge (generalizing some of its types if needed). The added edges are abstractions of the actions in A. To determine the frequency of the extended pattern we "join" the realizations of p with the realizations of the new added abstract action, deriving realizations for the extended pattern.

Note that the incremental nature of the algorithm matches nicely the incremental addition of new entity types (and their corresponding revision histories) in the refinement process described above, allowing the patterns to be gradually refined.

2.3 Using Windows and Patterns

We employ the discovered windows and patterns to correct Wikipedia entries as well as to assist users in editing.

Cleaning. One immediate application of the discovered patterns is to alert Wikipedia editors on partial edits performed in past windows. For that we examine the discovered windows and identify for each window and pattern (using an efficient outer-join based algorithm) maximal sets of actions that may be extended to a full pattern occurrence. Here again the different windows/patterns can be processed in parallel. Our UI, depicted in Figure 4, displays the

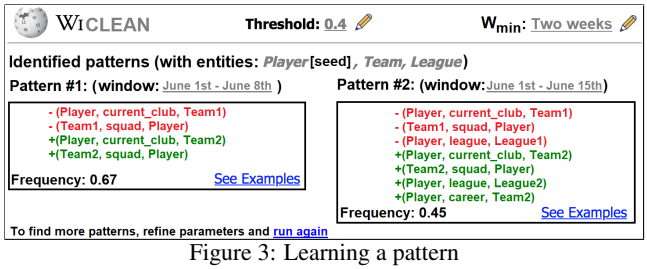

Figure 3: Learning a pattern

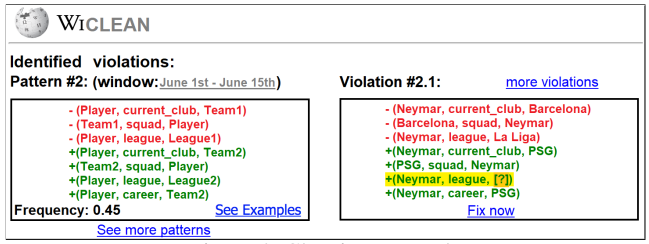

Figure 4: Cleaning Example

partial, instantiated pattern, highlighting the missing parts. We further present examples of other occurrences of the full pattern, to assist the editor in determining how (if) the partial edit should be completed (or alternatively reversed).

Edit assistance. Update patterns often appear periodically in multiple windows. For example the players transfer window repeats each summer with similar edit pattern. Our system automatically identifies such periodic patterns/windows and provides online edit assistance (through a plug-in) to users that update pattern entities within a given window, suggesting potential update completions, as explained above.

3. SYSTEM AND DEMONSTRATION

System. WICLEAN's implemented as a web browser extension, with backend in Python and frontend in JavaScript. The system employs three data stores: $Wikipedia\ Graph$ records all entities and links, $EditsDB$ records the entity revision histories and $PatternsDB$ records the generated patterns and windows. The users interact with the system via browser extension UI that allows users to control the frequency threshold τ and the width of the time window (in the pattern mining process), and to suggest edits (in the data cleaning/update process). The UI connects to the $Manager$ module that runs the two main system components: $PatternGenerator$ implements the mining algorithm, which takes the $Wikipedia\ Graph$ and the $EditsDB$ as input and stores the identified patterns and windows in the $PatternsDB$. *InconsistencyDetector* uses the patterns to signal updates that do not match the patterns and suggests possible corrections/completions. (We omit the architecture figure for space constraints.)

Demonstration. WICLEAN will be demonstrated on two entity types: Soccer players and Actors, assuming the audience will relate to at least one of them, then let users choose additional entity types as they wish. The demonstration will proceed as follows:

First, we will demonstrate how the system mines edit patterns and time frames, for the domain of choice, from real-life revision histories. The audience will observe the iterative pattern refinement process (as described in Section 2.2), and control the window size and the threshold, if desired, for more complex or relatively rarer patterns. An example of such interactions is depicted in Figure 3.

Next, we will demonstrate how the learned patterns are used for identifying incomplete/inconsistent past updates as well as for suggesting possible corrections (as described in Section 2.3). The users, playing the role of Wikipedia editors, will be presented with the suspected errors and correction recommendations. An example of such interactions is depicted in Figure 4. After completing this task (and actually making Wikipedia cleaner by fixing real-life errors), the users may continue editing other articles, using WICLEAN's recommendations for properly completing their edits.

To attract VLDB'19 attendees to our demonstration we will maintain a live scoreboard of users who have contributed most to cleaning and validating efforts. Furthermore, the contributors will be able to see how many potential readers their edits may reach.

Related Work. Our work is complementary to other Wikipedia cleaning techniques such as entity resolution, vandalism detection and others [9]. A closely related work [10] uses edit history of a knowledge base (Wikidata) to learn how to correct constraint violations. However they assume that the constraints are given and do not consider time windows where the constraints need not be enforced. Our work is complimentary, allowing to derive the constrains and the desired enforcement time. Another related work that focuses on infoboxes [4] predicts, based on past updates, when is a given infobox likely to be updated and who might be the editors, but does not consider the global pattern of the update nor related updates. Inconsistency detection has attracted much interest in the recent years [2, 3, 6, 8]. The detection often relies on the presence of integrity constraints, which may be viewed as a certain type of pattern. Automatic inference of integrity constraints has been extensively studied (see e.g. [5]). The difference in our setting is that the Wikipedia graph may be inconsistent at certain periods (inconsistency window), and these are precisely the points of interest in our case. Close to our work is research on pattern mining in sequence/temporal data [7], but the particularities of the Wikipedia settings make existing algorithms inapplicable as is. This includes the distribution of the revision history over a large number of individual entity logs (making it impractical to examine all logs in the mining process), the *connected* nature of the patterns we are seeking (rather than general frequent patterns), the frequent update reverts (which should be ignored), and the context dependent time windows (that need to be identified).

Acknowledgements This work has been partially funded by the Israel Innovation Authority, the Israel Science Foundation, Len Blavatnik and the Blavatnik Family foundation.

4. REFERENCES

[1] http://mappings.dbpedia.org/server/ontology/classes.html.
[2] A. Assadi, T. Milo, and S. Novgorodov. Cleaning data with constraints and experts. In *WebDB 2018*, pages 1:1–1:6.
[3] M. Bergman, T. Milo, S. Novgorodov, and W. C. Tan. Query-oriented data cleaning with oracles. In *SIGMOD*, pages 1199–1214, 2015.
[4] K. Bhatia, A. Halder, Y. Yadav, A. Sarsewar, P. Singh, and K. Khurana. Using big data technique for building edit alert system for wikipedia infoboxes based on map-reduce method. *IJIRCST*, 6:49–55, 07 2018.
[5] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.
[6] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, pages 1247–1261, 2015.
[7] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A survey of sequential pattern mining. 2017.
[8] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.
[9] A. Sarabadani, A. Halfaker, and D. Taraborelli. Building automated vandalism detection tools for wikidata. WWW 2017 Companion.
[10] T. P. Tanon, C. Bourgaux, and F. M. Suchanek. Learning how to correct a knowledge base from the edit history. In *WWW*, pages 1465–1475, 2019.
[11] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *SIGKDD*, pages 344–353, 2004.