



WeKnowIt

Emerging, Collective Intelligence for Personal,
Organisational and Social Use
FP7-215453

D3.2

Report on methods and tools for user feedback analysis

Dissemination level:	<Public>
Contractual date of delivery:	Month <18>, 2009-09-30
Actual date of delivery:	Month <18>, 2009-09-30
Workpackage:	WP3 Mass Intelligence
Task:	T3.2 Mass Interaction Feedback Analysis
Type:	<Report>
Approval Status:	< Draft PMB Final Draft Approved >
Version:	1.0
Number of pages:	24
Filename:	D3'2-v10'EMKA'deliverable.tex
Abstract	<p>This documents reports on new methods to analysis streams of user feedback data. The focus of this analysis are incremental clustering algorithms for graphs.</p> <p>The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.</p>



co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	2009-08-03	First draft	Michael Ovelgönne
0.2	2009-09-17	First version ready for internal review	Michael Ovelgönne, Andreas Geyer-Schulz
1.0	2009-09-30	Internal review comments incorporated (reviewer: Neil Ireson)	Michael Ovelgönne

Author list

Organization	Name	Contact Information
EMKA	Michael Ovelgönne	Phone: +49 721 608 5963 Fax: +49 721 608 8403 E-mail: michael.ovelgoenne@kit.edu
EMKA	Andreas Geyer-Schulz	Phone: +49 721 608 8402 Fax: +49 721 608 8403 E-mail: andreas.geyer-schulz@kit.edu

Executive Summary

This report on methods and tools for user feedback analysis starts with a detailed presentation and categorisation of user feedback data types and a discussion of the various kind of objectives we face when dealing with this data. Then, we present new methods to analyse feedback data and evaluate them. Finally, we show how this methods fit in the WeKnowIt Collective Intelligence approach and how the WeKnowIt use cases can benefit from this technology.

The overall objective of WeKnowIt is to develop technologies to exploit the distributed knowledge of large communities. The mentioned knowledge will be provided in various “encodings” (e.g. text, speech, still images, videos, ratings, relations), result of different kind of system usage (media upload as well as media download, ratings of media items), can be original knowledge (uploaded image) or derived knowledge (rating of a media item) and can be explicitly provided or can be implicit and just accessible through the observation of user actions.

Mass interaction feedback analysis aggregates the feedback of many users and derives its conclusions e.g. from structures or similarities in this data. This means, the analysis aims to remove the noise, i.e. the randomness. As a result we obtain results that the majority of feedback providing users implicitly or explicitly “agrees on”.

The WeKnowIt platform we develop will encourage users to interact with the system. From one user’s login to the system to his or her leave (a session) we can consider all interactions with the WeKnowIt system as a continuously flow of feedback. Even non-interaction is feedback. E.g. if a user views a page or not can be regarded as feedback on the page’s perceived utility.

As users continuously generate new feedback we have a steady increase in the amount of information we can exploit. Especially when the analysis should support emergency reponse, time is a very crucial issue. The stream of feedback will in some way reflect the course of events in an emergency situation and so should the analysis.

Abbreviations and Acronyms

RRW Restricted Random Walk

Table of Contents

1	Introduction.....	7
1.1	User Feedback Analysis.....	7
1.1.1	Bias in feedback data	7
1.1.2	Feedback analysis and the WeKnowIt system.....	8
1.2	Analysis of event streams	8
1.3	Related Work.....	9
2	Dynamic Graph Clustering	10
2.1	Graph clustering	10
2.1.1	Abstract problem description.....	10
2.2	Dynamic clustering algorithms	11
2.3	Stability vs. plasticity.....	11
2.4	Incremental restricted random walk clustering	12
2.4.1	RRW Clustering.....	12
2.4.2	Incremental update algorithm	13
3	Reference Implementation.....	15
3.1	API	15
3.2	Usage example	16
4	Performance test	17
5	Application scenario	19
5.1	Collective Intelligence scenario	19
5.2	Further application plans	21
6	Conclusion	22
7	References.....	23

List of Figures

1	Decreasing search space during restricted random walk	12
2	Service composition to yield a situation evolution monitoring service	20
3	Example of a tag cooccurrence network.	20

List of Tables

1 Introduction

1.1 User Feedback Analysis

When users interact with online or offline software applications we can - in the most broad sense - regard all user input as feedback of this user to the system. Even the lack of explicit user input can be regarded as a form of implicit feedback. As the interaction between user and system is an ongoing process and not a single point in time, users always generate streams of feedback data. And this streams of feedback data are our object of study.

Depending on the desired results, we can analyse explicitly provided feedback or derive implicit feedback from any system interaction. To give an example, imagine we are operating a travel website and want to provide recommendations for destinations “similar” to that one a user currently views. We could ask users for their recommendations and aggregate that data. Then, we have explicit feedback from users that we use to provide recommendations to other users. Another way to receive feedback would be to analyse the surfing behaviour of users. We can assume that web pages describing travel destinations that get viewed in one web session have something in common. A user browsing on the travel site searching for a destination for the next vacation will usually search for a specific type of destination (e.g. a trip to a Mediterranean beach or a city trip to Scandinavia) and will not randomly view web pages. Although the user does not provide explicit feedback on similar destinations, we get implicit feedback from observing the browsing behaviour.

The objective of mass user feedback analysis in general is to aggregate the feedback of many users (thousands, tens of thousands or even hundreds of thousands) and to find the common core of all provided data. Feedback data is usually not 100 % precise. It is full of noise - e.g. explicit feedback with human errors or implicit feedback where we wrongly interpreted the meaning of the observed behaviour.

1.1.1 Bias in feedback data

The feedback provided by users can be heavily biased by incentives of various kind. Implicit feedback is less affected by biases than explicit feedback. For feedback on the quality of a hotel there is a strong incentive for hotel owners to provide positive feedback for their own hotel and negative feedback for those of their direct competitors. To receive reliable results it is important to design the feedback system in a way that minimizes the expected bias. We could try to detect biased data and exclude it from the further analysis, but it is more promising to prevent biased feedback right from the start.

We want to emphasize that a scientific approach to the analysis of feedback starts way before developing algorithms. Especially because this report concentrates on the description of methods to find information in data it is important to note that these technologies are just one element of a successful system. Regulating the incentives to provide feedback is just as important.

Imagine a city wants to motivate its citizens to provide feedback during an emergency, even if they do not require immediate help, and promises a reward of 1 EUR for every submitted image of an ongoing emergency situation. Apparently, there is a motivation to upload as much images as possible - regardless of the seriousness of the observed situation. From this striking example

it is easy to see that wrong incentives can lead to an undesirable feedback behaviour. Many other less obvious incentives can result in distorted information input as well.

This short excursus should show that analysing the data is just half of the job. For an overview to the other half we refer to literature on mechanism design problems. A discussion of the mechanism design problems for recommender systems can be found in [16]. For the interpretation of clickthrough data see [13].

1.1.2 Feedback analysis and the WeKnowIt system

Analysing implicit feedback is very tempting because it does not require additional effort to generate the data. In the WeKnowIt system different kind of user input gets collected. Analysing this already available feedback means that more knowledge can be generated without the repeated problem of animating users to provide more input. And implicit feedback is usually less vulnerable for biases because users are not aware that they give feedback.

1.2 Analysis of event streams

Usually, when we analyse data the complete data set is known from the beginning. Either we have data with no time-reference or our data set comprises the collected data of some period of time in the past. However, when we need analysis results while new data is arriving or if we explicitly want to analyse the dynamics of the feedback data, we have to deal with event streams. This means we receive the data piece by piece and while we receive new pieces of information we continuously update our analysis. Especially for the emergency use case of WeKnowIt, we need to provide analysis results while the emergency is happening and cannot wait until it is over and the dataset is complete.

One way to deal with evolving data sets is to collect all events until a defined point in time and process this data. To update the results the old results will be revoked and the same methods as before will be applied to a dataset that contains all events until some later point in time. The drawback of this approach is that a lot of data has to be analysed repeatedly.

A second approach to analyse event streams is to perform incremental analysis. The updates will be generated just by adjusting the former results to the changes that have accumulated since the last update. If processing of all accumulated data takes more time than the desired update frequency we have to apply incremental update algorithms. Otherwise (when time and calculation costs are not critical) we can stick to the usually better studied and more established methods for the static data analysis.

As we want to be able to process a huge amount of data and provide a very short update frequency, so that we e.g. can provide the emergency response personnel with the very latest information as fast as possible we researched incremental update algorithms for the analysis of user feedback data.

We focused our research on processing streams of feedback data by means of cluster algorithms. Clustering is a very broadly applicable approach to data analysis. General clustering algorithms do not depend on tight assumptions about the nature of the input data and so we can use one technology in many different application scenarios.

1.3 Related Work

Analysing feedback data is a very broad field of research. For every kind of feedback and objective there is a research community searching for methods to find the desired information in that data. We restrict our general overview to some examples of feedback analysis and will provide an in-depth overview of related work for the focus technology of this report - incremental cluster update algorithms - in section 2.2.

Relevance feedback analysis (which used to improve the quality of information retrieval) has been researched for about 40 years. One of the early works has been published by [17]. Incremental algorithms have been developed e.g. by Aalbersberg [1] and Allan [3]. Although relevance feedback analysis is designed to work on the feedback of just one person, the latter works are of interest as they incrementally process feedback data. An example for using implicit relevance feedback data is the work of Agichtein et al. [2]. They analysed the user behaviour on a web search engine to improve the ranking of results. An overview of methods to infer the relevance of retrieved information from implicit feedback provides [14].

Most recommender systems work on feedback data, too. An example for a recommender system using explicit feedback is Grouplens [15]. One application of this collaborative filtering system is movielens (<http://www.movielens.org>). It uses the explicit ratings of movies to recommend to users movies that are rated best by other users with similar rating profiles. An example for a recommender system exploiting the implicit feedback of users from online library catalogs for book recommendations is described in [12].

2 Dynamic Graph Clustering

2.1 Graph clustering

A network representation of feedback data is very intuitive for several kinds of implicit and explicit user feedback as networks model relations between entities. This can be illustrated by the following example.

When users surf on a web site and view a series of pages we can assume that those pages are somehow related with regards to content. To reuse the example from the introduction, if some user browses on a travel site and views detailed description pages of a number of travel destinations, we can assume that these destinations have in some way a closer relation to each other than randomly picked ones (from the viewpoint of a particular user). Our network representation of page view behaviour is that all those pages are connected by a link that have been viewed in one user session. We receive a co-view network.

Usually, we have some randomness in our feedback data. In the aforementioned example, users could search for different types of destinations in one session or just have followed a link by mistake. When constructing a network we can aggregate the feedback of many users and weight the links between the entities of a network on basis of the aggregated feedback. By means of clustering we than are able to extract those connections we estimate to be non-random.

Clustering gives insight into the structure of networks by identifying “natural groups” of entities. The interpretation of the meaning of entities beeing in the same, respectively in different clusters depends on which information as been modeled in the analysed network.

2.1.1 Abstract problem description

A formal model for networks are graphs. A graph represent the entities and their relations by means of vertices respectively edges. We denote a graph as $G = (V, E, \omega)$, where $v_i \in V$ is a vertex, $e = (v_i, v_j) \in E$ a directed edge, $e = \{v_i, v_j\} \in E$ an undirected edge, ω_{ij} the weight of the edge connecting the vertices v_i and v_j . Henceforth, we just use edge weights greater or equal zero: $\omega : E \rightarrow \mathfrak{R}_0^+$.

Clustering means partitioning the vertices in groups of “closely related” vertices. We denote the set of clusters as $C = (c_1, \dots, c_n)$ and the assignment of vertices to clusters as $p : V \rightarrow C$, $p(v_x) = c_{y1}, \dots, c_{yn}$. There is no general definition of “closely related”. The usual requirement for good clustering is intra-cluster density and inter-cluster sparsity. That means, there are many connections between vertices within one cluster but few connections between vertices in different clusters.

Clustering can be overlapping (each vertex can belong to more than one cluster) or non-overlapping (each vertex belongs to exactly one cluster). Some fuzzy graph clustering methods measure for vertices the degree of membership to a specific cluster. The applicability of the various cluster methods depend on the desired result and on the available data.

2.2 Dynamic clustering algorithms

This section will give an overview of incremental cluster algorithms for dynamic (evolving) data sets. When a graph gets altered, we can distinguish two types of changes: changes in the set of vertices and changes in the set of edges respectively the edge weight function. Not every dynamic graph clustering algorithm is able to handle both types of changes.

Zhang et al. [20] developed the BIRCH algorithm, that can cluster data incrementally if additional refinement phases will be omitted. However, updating existing clusters produces errors that accumulate over time. Therefore, reclustering the complete data set once in a while is necessary to limit the amount of accumulated error. The idea of BIRCH is not to re-read vertices when updating the clusters but instead work on the clusters by representing them by a metric space.

Aslam et al. developed the star cluster algorithm [4, 5, 6]. Clusters consist of all vertices in the neighborhood of a cluster center. The clusters centers are the vertices with the highest degrees that are not already part of another cluster with a center with a higher degree. The algorithm is capable of incrementally updating the clustering on insertions and deletions of vertices and edges. One major downside of this method is the explicit structure of the clusters. Generally, star clusters will usually not reflect the intuitively expected groups in social networks.

Another algorithm for clustering dynamic datasets is the GRACE/GRIN algorithm by Chen et al. [7, 8]. The algorithm is based on an analogy to gravity. The method is only capable to handle insertions. The GRIN algorithm assigns new objects to existing clusters if this complies with the spherical shape requirement. Otherwise, clusters need to be resized or objects will be declared as outliers. Regular reclustering is an explicit part of this algorithm. If the number of outliers that could not be assigned to any cluster exceeded a threshold, the GRACE algorithm is used to recluster the data.

An incremental version of the widely known DBSCAN algorithm [18] has been developed by Ester et al. [10]. The idea of DBSCAN is to define clusters as regions with high local density. Objects within one cluster must be directly density-reachable or reachable via a chain of intermediate objects. One object is density-reachable by another object if it is within a specific radius and a minimum number of other objects are also in this area. The incremental DBSCAN algorithm support insertions and deletions and exploits the locality of the density-reachable function.

2.3 Stability vs. plasticity

Duda et al. [9] discussed the important issue of the diverging objectives of stability and plasticity. In this context, stability means that clusters remain unchanged when the changes to the underlying graph are minor. In contrast, plasticity means that the clustering should reflect the available data as accurately as possible and should not depend improperly on previous states of the graph.

Of course, there is a tradeoff between stability and plasticity. The more stable a clustering is the less accurate it can adapt to changes in the data set. Clearly, the optimal balance between stability and plasticity depends on the specific application scenario.

2.4 Incremental restricted random walk clustering

2.4.1 RRW Clustering

Restricted random walk clustering takes directed or undirected graphs with a similarity function as an input. An edge weight function assigns some kind of “costs” to edges that arise when the edge is passed. For example, the natural edge weight in a highway network would be distance. On the contrary, a similarity function does not describe distances (or costs etc.) but proximity.

Restricted random walk clustering works with so-called walks. A walk is a series of vertices that is constructed as follows: Starting at some initial vertex the next vertex in row is chosen randomly among all connected vertices that are reachable via an edge whose weight is higher than the weight of the edge used to get to the actual vertex. The walk ends at the point where no adjacent vertex meets this requirement (see 1). During the walk the weights of the edges used to walk through the graph always increase. As the edge weight should be a measure for the similarity (not the distance!) of the adjacent vertices the vertices at the end of the walk are more similar than those at the beginning. Accordingly, succeeding vertices at a high step number in a walk are very likely to be much more related than other vertices in their neighborhood. The longer a walk the more reliable is the inherent information on the relatedness of succeeding vertices.

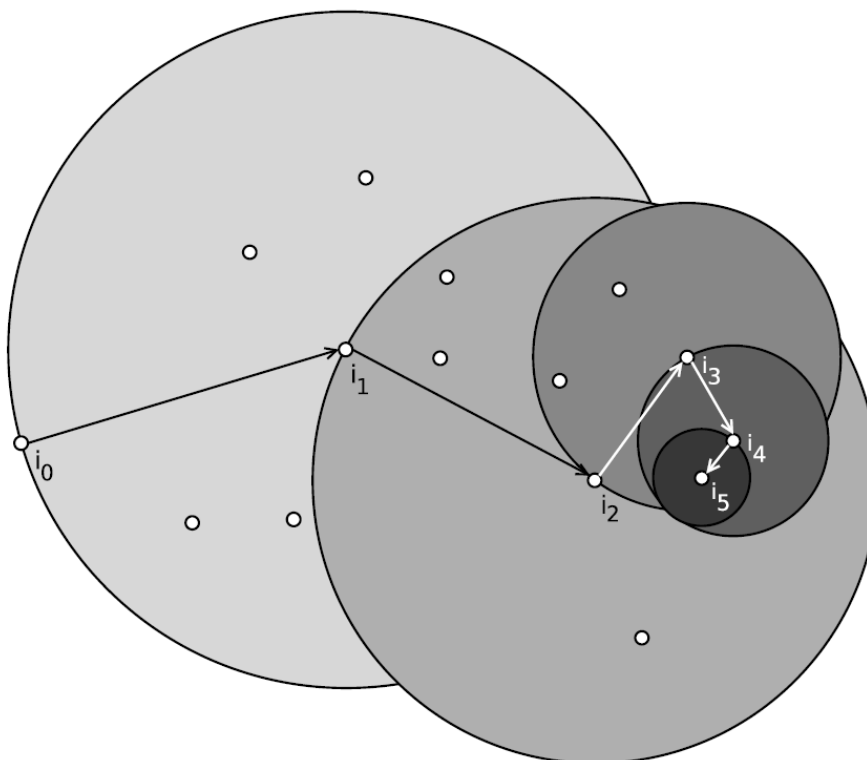


Figure 1: Decreasing search space during restricted random walk

To explore the structure of the graph, k random walks start at every vertex. Clusters can be extracted from the walks by a method called *walk context clusters* [11]. A cutoff level l is defined to disregard vertices that appear at the beginning of a walk where randomness has greater influence. For a given vertex v_i all vertices that appear after step c in any walk where also v_i appears after step c are regarded as a cluster. A uniform value for the cutoff level l causes

problems in graphs with sections of varying walk length. Therefore, l should be adapted to the local walk situation by using $l = (\text{step number}) / (\text{walk length} + 1)$.

The walk context clusters are overlapping. An alternative approach produces non-overlapping clusters [19]. From the original graph $G = (V, E)$ a series of subgraphs $G_k = (V, E_k)$ are created that contain all edges that have been used in the k -th step of any walk. The union of all graphs G_k with a k greater or equal a level parameter l gives a second series of graphs $H_k = \bigcup_{l=k}^{\infty} G_l$. The connected components of H_k are the clusters for the parameter k .

The component clusters have the disadvantage that they produce large clusters if some bridge vertex connects otherwise marginally connected groups of vertices. The walk context clusters do not tend to bridging as much as component clusters.

2.4.2 Incremental update algorithm

In the last section we described how clusters can be generated from random walks on a static graph. Next, we show how to incorporate insertions, deletions and changes in the similarity matrix.

Changes in the similarity matrix can lead to two situations: 1. A random walk can follow new paths 2. Some once legal successors become illegal. In the first case the update works as follows: We need to identify all transitions (edges) that after the similarity change have at least one new possible successor. We denote the weight function after the update as w' . The set of new possible successors of an edge (v_i, v_j) is given by the following formula:

$$s_{ij} = \{(v_j, v_k) | w'_{jk} > w'_{ij} \wedge w_{jk} \leq w_{ij}\} \tag{2.1}$$

For all walks that contain an edge (v_i, v_j) with $s_{ij} \neq \emptyset$ we need to adapt the walks to receive a probability distribution of the walks that we would have received if the change in the similarity matrix would have taken place before the determination of the original walk. If we denote T_{ij} as the set of possible successors of (v_i, v_j) before the similarity update and T'_{ij} as the set of possible successors after the update then an affected walk needs to be truncated after the transition from i to j and continued by one of the new possible successors with a probability of $P(\text{change}) = 1 - \frac{|T_{ij}|}{|T'_{ij}|}$.

In the second case, when certain transitions become illegal, we once again need to find all affected walks and update them. The walk will be truncated before the illegal successor and the original algorithm can be used to continue the walk on another path. Successors become illegal when they no longer have a higher weight as the preceding transition. If an edge weight has been increased, a potential cut has to be applied after that edge. In case a edge weight was decreased a walk might have to be cut before this edge.

Beside handling the change of edge weights, insertions and deletions of vertices have to be processed. For every new vertex we need to start a specified number of walks at this vertex using the original algorithm. Additionally, the edges of the new vertex have to be updated as if they were edge weight changes. Deletion of vertices are handled in a similar way. The walks starting at this vertex will be deleted and all walks containing the deleted vertex must be updated as described for illegal successors.

This update procedure results in the same probability distribution for the random walks as the original algorithm with the additional information would have resulted in. That means, updates do not produce errors and no complete re-clustering is needed to keep the effect of accumulated errors low. For the proof of the equivalence of the probability distribution of the update algorithm to the original algorithm see [11].

This incremental update algorithm is the first fast update method for graph clustering that supports insertions and deletions of vertices and edges and works without regularly reclustering all accumulated data.

3 Reference Implementation

For the evaluation of the algorithm we have written a reference implementation. The two main requirements for this implementation were a high update performance and the ability to work on very large graphs. To fulfill the second requirement, keeping the data in main memory was no option. The main memory has a very low size compared to other forms of data storage (hard disk drives, solid state drives). Therefore, the reference implementation operates directly on a relational database. The database management system (DBMS) in our test environment is PostgreSQL 8.3. This DBMS is able to manage table sizes up to 32TB. For the intended usage we will not be restricted by storage limits. However, access times and transfer rates are significantly lower than those of main memory. The operating system caching mechanisms ease this problem. So, available main memory increases the performance but does not limit the size of the network we are able to cluster. The skillful choice of indexes for the database tables is essential for a good system performance.

Our implementation consists of two main methods. The initial walker mechanism and the update mechanism. The initial walker creates the restricted random walks for the first time. This mechanism starts at each vertex a predefined number of walks and stores them in the walk table of the database. The update mechanism reads the changes in the graph from a delta table, updates the graph table and updates the walks from the walk table if necessary.

The delta table operates as a cache to store all changes until the next update. The decision when to update the clustering depends on the requirements of the application calling the update mechanism. In some application scenarios nightly updates might be appropriate to minimize the database load during daytime. In other scenarios it might be best to run updates after every change in the data set to have the most up-to-date clustering at any point in time.

The walk context clusters can be extracted as they are needed. We are dealing with clusters from the perspective of vertices. That means, for every vertex we generate a separate set of vertices that are regarded as the cluster of the particular vertex. Constructing the cluster for a single vertex is fast. Therefore, we do not need to update all prospective clusters for all vertices after updating a vertex if we do not need them. According to the application scenario we can restrict this to the vertices of interest.

3.1 API

For the further usage within the WeKnowIt use case implementations the restricted random walk clustering algorithms can be accessed via an API. All functionality is bundle in the class `RRWClustering`.

The constructor of `RRWClustering` creates a connection to the database. Therefore, server name, database name and login credentials must be provided. Users of the API must provide credentials that have sufficient rights for all database operations invoked by any method of `RRWClustering` called latter on.

Three methods are provided to store the graph. While `addVertex` can be used at any time to insert a new vertex into the graph, `addEdge` may only be used to insert edges before the initial calculation of the clustering. After the initialization the edge weights must be updated using `updateEdge`. `UpdateEdge` must be used to insert a new edge after the initialization as well. This

methods stores the changes in a delta table and the graph and the clustering will not be changed until update is called.

The methods `init` and `update` are used to initially calculate the clustering respectively to update the clustering after changes of the graph. `update` processes all changes since its last call.

The method `getSameCluster` return all other vertices in the same cluster as the vertex with the given ID. The parameter `minLevel` has to be a value between 0 and 10. It is the level parameter l of the walk context cluster algorithm we use in this method.

The last two methods are provided for convenience. The method `createDatabaseTables` can be used to create all necessary database table. Existing tables with the same names as the ones to create will be dropped. The second methods, `clearAllTable`, just deletes all data from all tables used by the `rrw` implementation. This methods require that the provided login credentials have sufficient rights for this operations.

3.2 Usage example

The following example shows how the API can be used to cluster a network and update the clustering after a change in the graph. This example presumes that there is a method `printSameCluster` that prints all elements of a list to the standard output. Then, this example creates a new instance of `RRWClustering` and fills it with initial data, clusters the graph and print out all vertices that are in the same cluster as vertex 1. Afterwards, the graph is altered, the clustering is updated and the all vertices that are now in the same cluster as vertex 1 are printed to the screen.

```
public static void main(String[] args) {
    RRWClustering rrw = new RRWClustering("postgresql.weknowit.eu",
        "database_1", "user_1", "password");
    rrw.createDatabaseTables();

    /* Create random network */
    rrw.addVertex(1);
    rrw.addVertex(2);
    rrw.addVertex(3);
    rrw.addVertex(4);
    rrw.addVertex(5);

    rrw.addEdge(1, 2, 1);
    rrw.addEdge(3, 2, 2);
    rrw.addEdge(4, 3, 1);
    rrw.addEdge(4, 5, 1);

    /* Cluster network */
    rrw.init();
    printSameCluster(rrw.getSameCluster(1), 5);

    /* Change network */
    rrw.updateEdge(4, 3, 2);
    rrw.updateEdge(1, 5, 3);

    /* Update clusters */
    rrw.update();

    printSameCluster(rrw.getSameCluster(1), 5);
}
```

4 Performance test

Performance is critical for processing feedback streams. For the evaluation of the run time in a life setting it is important to notice that not every update also forces to update the walks. Therefore, not every update changes clusters. The run-time of incorporating a single update differs heavily depending on the need to update a walk.

We set up a test to determine the update time for the incremental cluster algorithm. To test how the algorithm performs under very challenging situations, we used the largest dataset we could obtain. For that reason, we took a data set from the University of Karlsruhe library OPAC (online public access catalog). This data set consists of roughly 500.000 vertices and 20 million edges. In this data set vertices are library items (usually books) and edges indicate that a OPAC user visited the detail pages of both adjacent books in one online session. The type of data is e.g. what we would receive if we log user behavior on a travel site. We could cluster those data to identify similar travel destinations and build a recommendation service with this results.

We added to the initial set of feedback data (consisting of seven years of OPAC user data) chunks of additional data (each consisting of the feedback of one month). On average, only 3.17% of the similarity changes required a walk update and our implementation was able to process a single update in 155 msec. That means, we can process several user feedback updates per second. This is a sufficient capacity even for emergency situations with sudden peaks in user activity.

One circumstance of our studied test scenario contributes to the good performance of the algo-

rithm: If we have a low number of accumulated feedback updates to process, the probability that a new feedback forces to update walks is high. But if we already have a lot of feedback and the similarity graph is large, each additional feedback triggers a walk update only with a low probability. Altogether, while the costs for updating walks and re-creating clusters increases with the number of feedback units, the probability that this operation is necessary decreases.

5 Application scenario

The application of mass user feedback analysis for generating destination recommendations on travel websites has already been mentioned in the first section. This application idea refers directly to the user requirements analysed in the deliverable D7.2 (ER/CSG case study design and specification). To show the significance of the developed technology to the Collective Intelligence approach of WeKnowIt, we will present another application scenario where Media Intelligence will be enriched by Mass Intelligence.

5.1 Collective Intelligence scenario

In the emergency response use case of WeKnowIt the developed services and the system as a whole shall support emergency response personnel and the public by providing information of various kinds. This information shall empower officials and citizens to appropriately react on the emergency situation. From an improved information situation all sorts of possible sources of error can benefit: effort allocation, collaboration and general decision making.

During a major emergency the situation will usually constantly keep changing. One threat gets resolved or maybe vanishes by sheer chance while another threat arises. During a flood an endangered dam might break, be repaired by local residents or the threat may disappear as the water pressure decreases because of a dam break at some other place or less replenishing water. For emergency response personnel it is very important to have a reliable and up-to-date picture of the situation. Available personnel and equipment is mostly scarce so that operations planning can not afford sending relief units to no longer existing risk situations.

One component of a system for the aggregation of the current emergency situation could be a module for the automated analysis of user media submissions. The WeKnowIt system will support uploading media items (image, audio, video). Users can provide other users or emergency response staff with this information. Additionally to the manual utilization of this information we propose a service that automatically analyses the data for identifying upcoming changes in the situation. For this service the mass feedback analysis will take tagged media items as an input. Images, videos or audio files can be annotated with tags so that beside the pure media content there are additional meta data available describing the content of the media. From experience we know that users annotate uploaded media only partially and incompletely. Therefore, in WP2 a service will be developed for automated tagging of media. Because of this service, all media uploaded to the WeKnowIt system will be annotated with tags at the latest after having been processed by the media tagging service.

From the stream of user uploads we can analyse the derived stream of tags. When one media item is annotated with more than one tag we can assume that these tags have some kind of relation. The tags will probably describe a place, an event and/or an effect. From this cooccurrence knowledge we can build a graph of tag cooccurrences. For an example see figure 3. With every new uploaded and tagged media item we get new feedback that we incorporate into the cooccurrence graph (thick line). By incrementally updating the clusters we know at any point in time which tags are closely related. If in our example “passable” is new in the cluster of “High street” after a cluster update this is an indication that the situation “flooded” (which got connected with “High street” at some time before) changed.

For an emergency response officer the knowledge about which tags are associated is too detailed

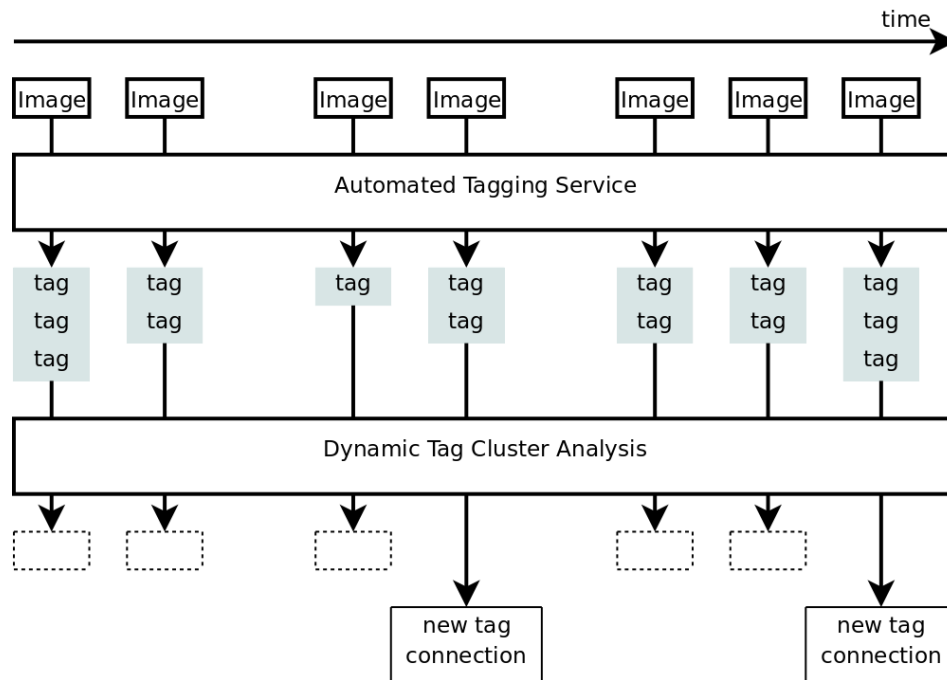


Figure 2: Service composition to yield a situation evolution monitoring service

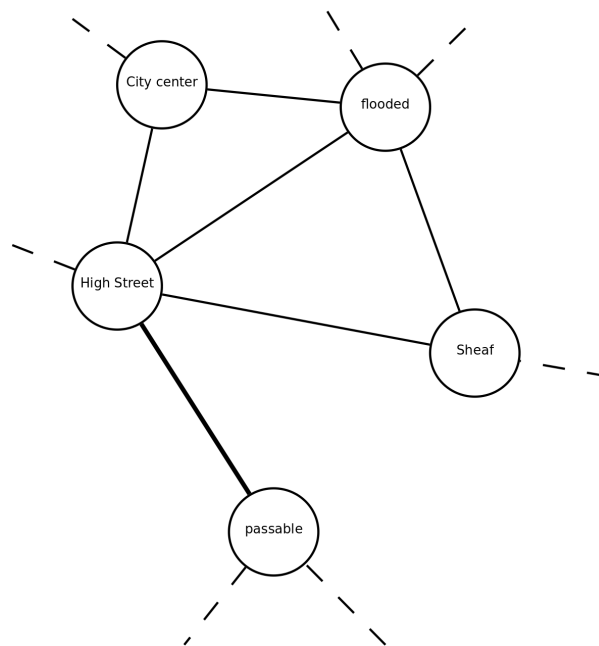


Figure 3: Example of a tag cooccurrence network.

of information to be of any practical value. But if we list the cluster changes after each update we are able to restrict the amount of information to what is important. Changes in the relation of tags will reflect a change in the situation. Therefore, we are able to detect new situations as soon as a significant amount of media recording that situation has been uploaded.

5.2 Further application plans

The fast cluster update technology can be exploited in task T3.4 for mass evolution analysis, too. Analysing the evolution of data means for example the identification of trends. The walk context cluster approach to generate clusters from restricted random walks provides information on how strong a vertex belongs to the cluster of another vertex. The evolution of this information indicates how the relation of two vertices changes over time.

6 Conclusion

In this deliverable we presented new methods of user feedback analysis. We put the emphasis on the analysis of streams of user feedback data. To provide analysis results as fast as possible feedback data has to be analysed as it gets generated. Usually, the most efficient way to analyse data streams are incremental update algorithms. A general purpose method for data analysis is clustering. Clustering graphs provides insight to the structure of the relations between the modeled entities. The presented incremental update algorithm for restricted random walk clusters is able to integrate new information on the relationships of entities as they get available. With this technology all sorts of streams of data that contain information entities and their relations can be processed efficiently.

As an example of an application scenario we presented an scenario for feedback analysis methods in an emergency response use case to demonstrate the applicability of the methods to WeKnowIt. The value we believe this service bundle could provide, results from its Collective Intelligence approach. Only the smart combination of different intelligence layers is able to provide valueable and ready-to-use information without the need of human intervention. We showed, how WeKnowIt Mass Intelligence technology can be used to provide emergency response teams with instant updates on changes of the emergency situation by automatically exploiting incoming tag streams. Those tag streams can be generated by the automatted tagging service of the Media Intelligence layer that processed streams of incoming media items (image, video, audio). Those media tags will be analysed to identify closely interrelated tags. As new interrelations are likely to be caused by changes in the emergency situation, the tag cluster analysis could support emergency officers to keep track of the situation.

7 References

- [1] IJsbrand Jan Aalbersberg. Incremental relevance feedback. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 11–22, New York, NY, USA, 1992. ACM.
- [2] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26, New York, NY, USA, 2006. ACM.
- [3] James Allan. Incremental relevance feedback for information filtering. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 270–278, New York, NY, USA, 1996. ACM.
- [4] Javed Aslam, Katya Pelehov, and Daniela Rus. Computing dense clusters on-line for information organization. Technical Report PCS-TR97-324, 1997.
- [5] Javed Aslam, Katya Pelehov, and Daniela Rus. Static and dynamic information organization with star clusters. In Georges Gardarin, James C. French, Niki Pissinou, Kia Makki, and Luc Bouganim, editors, *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management*, pages 208 – 217, 1998.
- [6] Javed Aslam, Katya Pelehov, and Daniela Rus. A practical clustering algorithm for static and dynamic information organization. In Robert E. Tarjan and Tandy Warnow, editors, *SODA '99*, pages 51 – 60, Philadelphia, 1999.
- [7] Chien-Yu Chen, Shien-Ching Hwang, and Yen-Jen Oyang. An incremental hierarchical data clustering algorithm based on gravity theory. In M.S. Chen, P.S. Yu, and B liu, editors, *Advances in Knowledge Discovery and Data Mining*, volume 2336 of *Lecture Notes in Computer Science*, pages 237 – 250, Heidelberg, 2002. Springer.
- [8] Chien-Yu Chen, Shien-Ching Hwang, and Yen-Jen Oyang. A statistics-based approach to control the quality of subclusters in incremental gravitational clustering. *Pattern Recognition*, 38(12):2256 – 2269, 2005.
- [9] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, New York, 2 edition, 2001.
- [10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 323 – 333, San Francisco, 1998. Morgan Kaufmann Publishers Inc.
- [11] Markus Franke and Andreas Geyer-Schulz. An update algorithm for restricted random walk clustering for dynamic data sets. *Advances in Data Analysis and Classification*, 3(1):63 – 92, 2009.
- [12] Andreas Geyer-Schulz, Andreas Neumann, and Anke Thede. Others also use. In Traugott Koch and Ingeborg Torvik Solvberg, editors, *Research and Advanced Technology for Digital Libraries*, volume 2769 of *LNCS*, pages 113 – 125, Berlin, 2003. Springer.

- [13] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 154–161, New York, NY, USA, 2005. ACM.
- [14] Diane Kelly and Jaime Teevan. Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum*, 37(2):18–28, 2003.
- [15] Joseph Konstan, Bradley Miller, David Maltz, Jonathan Herlocker, Lee Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usernet news. *Communications of the ACM*, 40(3):77 – 87, 3 1997.
- [16] Andreas W. Neumann. Motivating and supporting user interaction with recommender systems. In László Kovács, Norbert Fuhr, and Carlo Meghini, editors, *Research and Advanced Technology for Digital Libraries*, LNCS 4675, pages 428–439, Berlin Heidelberg New York, 2007. Springer.
- [17] J.J. Rocchio. *Relevance Feedback in Information Retrieval*, pages 313– 323. Prentice-Hall, 1971.
- [18] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, 2(2):169 – 194, 1998.
- [19] Joachim Schöll and Elisabeth Paschinger. Cluster analysis with restricted random walks. In Krzysztof Jajuga, Andrzej Sokolowski, and Hans-Hermann Bock, editors, *Classification, Clustering, and Data Analysis*, volume 21 of *Studies in Classification, Data Analysis, and Knowledge Organization*, pages 113 – 120, Heidelberg, 2002. Springer-Verlag.
- [20] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch. In H. V. Jagadish and Inderalp Singh Mumick, editors, *Proceedings of ACM SIGMOD international conference on Management of data*, volume 25, pages 103 – 114, Montreal, Quebec, Canada, 1996. ACM Press.