SOCLE: Towards a Framework for Data Preparation in Social Applications

Sihem Amer-Yahia [1],
Noha Ibrahim, Christiane Kamdem Kengne [2],
Federico Ulliana, Marie-Christine Rousset [2]

1. *CNRS, LIG, Grenoble, France*
   Sihem.Amer-Yahia@imag.fr
2. *University of Grenoble, LIG, Grenoble, France*
   firstname.name@imag.fr

ABSTRACT. *The web has evolved from a technological platform to a real social milieu thereby becoming a continuous source of Big Social Data (BSD). BSD is characterized by a combination of factual content such as the coordinates of a restaurant, the content of a webpage or the title of a movie, behavior data such as exchanges between social relationships, as well as subjective data such as users' opinions, reviews, and tags. The goal of a social application is to analyze BSD and process it in order to understand it and transform it into valuable content to users. Building social applications requires an essential* data preparation *step during which raw BSD is sanitized, normalized, enriched, pruned, and transformed making it readily available for further processing. We argue for the need to formalize data preparation and develop appropriate tools to enable easy prototyping of social applications. We describe* SOCLE*, our framework for BSD preparation. We provide an architecture inspired from typical social applications, the state of the art of existing languages and algebras for manipulating BSD, and the scientific challenges and opportunities underlying the development of* SOCLE.

RÉSUMÉ. *Le web, initialement une plateforme technologique, est devenu un véritable éco-système social et une source continue de Big Data sociales (BSD). Nous y trouvons des données factuelles et objectives telles que les coordonnées d'un restaurant, des données comportementales telles que les échanges entre amis et enfin, des données subjectives comme les revues, les étiquettes, les notes ou les étoiles. Ces données brutes ont besoin d'être filtrées et organisées pour en extraire des informations utiles et permettre le développement d'applications sociales qui apportent une valeur ajoutée aux utilisateurs. Dans cet article, nous motivons le besoin de formaliser l'étape de* préparation des BSD *et décrivons* SOCLE*, notre plateforme pour le faire. Nous présentons une architecture inspirée d'applications sociales types, un état de l'art des langages et algèbres existants et les défis scientifiques du développement de* SOCLE.

KEYWORDS: *Big Social Data (BSD), Social Applications, Social Graph, Algebra*

MOTS-CLÉS : *Applications Sociales, Graphe Social, Algèbre, Langage Déclaratif*

DOI:10.3166/ISI.volume.issue.1-24

1. Introduction

The survival of the social web relies heavily on the development of social applications whose goal is to analyze social data and deliver relevant content to users thereby providing value and incentives on the social Web. Researchers and practitioners have been actively building a variety of social applications such as website recommendation on del.icio.us, sentiment extraction from Twitter, movie recommendation on MovieLens and itinerary extraction from Flickr. *Data preparation*, that is the step of pre-processing raw input datasets into ready-to-be-exploited data, is an essential step in building those applications. It is however proprietary to each implementation and mostly "encapsulated" in scripts that are hard to verify and modify. In this paper, we present SOCLE, the first framework for expressing and optimizing data preparation in social applications.

1.1. Social Data and Applications

The web has evolved from a technological platform to a real social milieu and is becoming a continuous source of Big Social Data (BSD). BSD is characterized by a combination of factual content such as the coordinates of a restaurant or the content of a webpage, behavior data such as the exchanges between social relationships, as well as subjective data such as users' ratings, reviews, and tags. The very first step in building social applications is data preparation that is common to all those applications and that is repeatedly performed by data scientists today. The large volumes of raw BSD call for the development of scalable models and algorithms to prepare such datasets. In this paper, we make a first attempt at formalizing SOCLE a generic and extensible framework for raw BSD preparation.

Data preparation in building social applications is an essential step during which raw BSD is sanitized, normalized, enriched, pruned, and transformed to be made readily available for developing a given application semantics. For example, for website recommendation on del.icio.us, users' tagging actions are pre-processed in a preliminary step, in order to remove the long tail of tagging, extract topics from tags and cluster users together (Maniu, Cautis, 2012; Schenkel *et al.*, 2008; Stoyanovich *et al.*, 2008). Similarly, in a movie recommendation application on MovieLens that caters to user groups, users' ratings are normalized and users are pre-grouped based on rating similarities in order to optimize the retrieval of movies for arbitrary user groups (Amer-Yahia, Roy *et al.*, 2009; Schafer *et al.*, 2007; O'Connor *et al.*, 2001). When doing sentiment extraction on Twitter, tweets are pre-processed to extract topics and entities (Davidov *et al.*, 2010). In news recommendation (Abbar *et al.*, 2013), articles are pre-processed to extract topics and sentiment extraction is applied to user comments on articles (Amer-Yahia *et al.*, 2012). Finally, when Flickr photos are used to build touristic itineraries in a city (De Choudhury *et al.*, 2010), tags and geographic coordinates are used to map individual photos to landmarks thereby enriching raw social data. All those are examples of primitive data preparation operations that are repeatedly hard-coded in various applications. **Our first contribution (Section 2) is**

the study of typical social applications and the design of a general-purpose architecture for BSD, that identifies the main phases of BSD preparation, namely *Data Collection*, *Data Sanitization*, and *Data Transformation*. We proceed in a bottom-up fashion starting from typical social applications to design *a generic and extensible framework* for expressing the main phases of BSD preparation.

1.2. Benefits

The sheer volume of BSD and the many different alternative semantics in building social applications would benefit from the development of a declarative approach to enable flexible prototyping. That is particularly true for data preparation in which many "choices" are made that later affect the results of a search and recommendation application. More precisely, content usefulness in search and recommendation applications is typically defined using application-dependent semantics under the form of utility functions. A variety of those functions are implemented ranging from overall popularity of a content item such as a restaurant, a website or a movie to personalized functions such as websites endorsed by friends of a user. The success of an application relies heavily on the design of appropriate utility functions. That is why an essential step in building social applications is the evaluation of the utility of returned content.

User studies on platforms such as Amazon Mechanical Turk or CrowdFlower combined with more traditional techniques from Machine Learning such as A/B testing, or from Information Retrieval such as precision and recall, are used to perform the evaluation. When the results of an evaluation are unsatisfactory, application developers tend to "blame" the definition of utility and explore ways to fine-tune the utility function. For example, if the use of tag-based similarity networks on del.icio.us is not a good predictor of future user tagging actions, an application developer will experiment with other networks such as tag and URL similarities. As a result, application developers modify the application semantics and rebuild a different logic from scratch and run the evaluation again. The other data preparation steps that were undertaken are often ignored because they tend to be encapsulated in code and are therefore "invisible". What if unsatisfactory results were due to thresholds in cutting the tail of tagging, or to the method used to extract topics from tags or sentiment from reviews?

Finally, although the development of SOCLE follows a bottom-up approach using a small number of social applications, we believe its extensible and generic design enables the expression of a wider set of social applications. SOCLE is a start toward that direction and it will not impede the integration of new data manipulation operations at any one of its three key levels: Collection, Sanitization and Transformation.

1.3. Paper Organization

We argue that the evaluation step in building social applications should be rethought to enable *a feedback loop into all data preparation steps*. Therefore, **we propose a declarative approach to data preparation and explore the use of an**

algebra to express the main operations identified in the SOCLE architecture. The choice of an algebra is motivated by the need for a declarative framework that enables optimization. Indeed, while workflows and process algebras can capture task coordination appropriately, they do not express data transformations explicitly and are not well-suited for capturing data preparation.

No algebra is possible without a data model. In Section 3, we start with a review of existing models for representing BSD and discuss the requirements and opportunities in designing an appropriate model to capture the variety of social data. In particular, we describe the expressivity of existing proposals and discuss one major question, that is, storing BSD. We then present the SOCLE data model. There exists many algebras and languages for manipulating social data. Section 4 examines existing algebras and languages for querying and transforming BSD as well as other data and process manipulation languages, and discusses new opportunities raised by the requirements of data preparation. In particular, the ability to undo some steps that were undertaken in data preparation and the optimization opportunities this requirement raises. The SOCLE algebra is presented in this section using examples inspired from the social applications described in Section 2.2. Finally, in Section 5, we describe our immediate plan of action for SOCLE, that is, the implementation of a library of data preparation tools and its validation in a movie recommendation application scenario. More specifically, **we are exploring the use of native and non-native graph storage options and the implementation of primitives in the SOCLE algebra on top of those two backends.**

In the rest of the paper, we use the term SOCLE interchangeably to designate the framework, the architecture, the algebra and language.

2. Architecture and Applications

In this section, we present the architecture of SOCLE, our BSD preparation framework. We examine four examples of real-world applications on social content sites and describe how each one "instantiates" our architecture.

2.1. Architecture

The social data we are interested in is in the form of $\langle u, i, l\rangle$ triples that designate the action of a user u in a set of users $\mathcal{U}$ on an item i in a set of items $\mathcal{I}$ with a label l in a set of labels $\mathcal{L}$ that includes tags, ratings, reviews, comments, or sentiments depending on the underlying data. For example, $\langle John, foodnsport, sports\rangle$ and $\langle John, foodnsport, diet\rangle$, in del.icio.us, represent user John who tagged the foodnsport website with the tags sports and diet. In MovieLens, $\langle Sue, Titanic, 5\rangle$ represents user Sue who rated the movie Titanic with the highest rating. On Twitter, the triple $\langle John, Obama, pos\rangle$ expresses that user John's sentiment on entity Obama is positive. Finally, in Flickr, $\langle Rob, 534, Louvre\rangle$, $\langle Rob, 534, Paris\rangle$ represent that user Rob tagged the same picture with Louvre and with Paris.

We examined a large number of social applications [1]. Three layers that form BSD preparation emerged from that: *Data Collection*, *Data Sanitization* and *Data Transformation*. They are depicted in the logical architecture of SOCLE (Figure 1). All the applications we examined gather and integrate data, possibly coming from different sources. Then, data is normalized, pruned and enriched. Finally, several transformations are necessary to extract value from sanitized data and optimize its usage. The result of data preparation is ready-to-be exploited data that is fed to the *Social Application Logic* box which builds a variety of social applications ranging from website and movie recommendation to itinerary extraction on Flickr and social search on Twitter.

In summary, BSD preparation admits $\langle u, i, l \rangle$ triples and returns two types of data: processed data that was collected and sanitized, and transformed data (e.g., per user networks of interest). The returned data is then used by the *Social Application Logic* component to implement a search and recommendation or an analytics application.

Data Collection. This layer gathers and integrates social data from different sites. This data is available under different forms, either as a dump (e.g., dump of ASCII files from MovieLens or XML dump from DBLP), or via an API (e.g., Flickr or Twitter API). Gathered data needs to be "wrapped" into our $\langle u, i, l \rangle$ triples model. Although there is little potential for generalizing this step, it could be automated by providing customized wrappers for existing sites such as Flickr, del.icio.us, and Twitter.

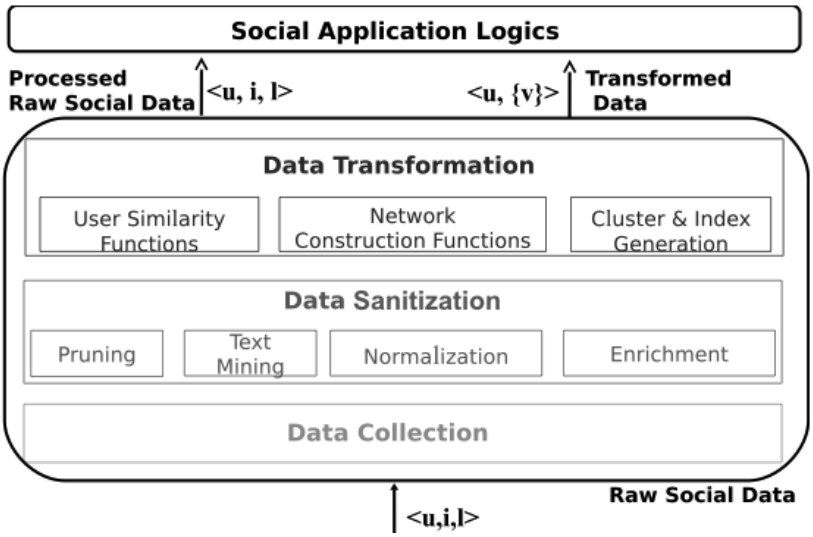


Figure 1. The SOCLE *architecture*

Data Sanitization. This layer takes as input all the raw data collected by the *Data Collection* module and applies different operations in order to prepare it for transfor-

1. We read over 50 papers and browsed multiple websites. In particular, the experiments section of each paper contained different descriptions of how data is pre-processed

mation. We grouped those operations into four components: *Pruning*, *Text Mining*, *Normalization* and *Enrichment*. The *Pruning* component removes all unnecessary information such as the long tail of tagging actions (e.g., uncommon tags or items that were tagged by less than a certain number of users, photos that are not relevant for itinerary extraction). As a result, the dataset is reduced and social applications do not have to deal with irrelevant information. The *Text Mining* component extracts valuable information from textual content such as tweets, reviews and comments. Examples of such operations include topic detection from tags, entity and event identification from tweets, and sentiment extraction from reviews. The *Normalization* component applies value transformation operations such as restaurant or movie rating normalization. The *Enrichment* component adds semantics to data by using external sources. For example, this component is used to detect when users tag similar photos with different namestags (e.g., "NYC", "Manhattan", "The Big Apple", all relate to New York city). In this case, the *Enrichment* component adds a number of predefined tags found in an external source such as guidebooks.

Data Transformation. While *Data Collection* and *Data Sanitization* manipulate data in the form of $\langle u, i, l \rangle$ triples, *Data Transformation* modifies data and may alter its form. The *Normalization* component in *Data Sanitization* may modify rating values but it does not modify the $\langle u, i, l \rangle$ triple form. *Data Transformation* has three main components. The *User Similarity* component is central to search and recommendation. It is responsible for computing similarities between users. In a given social application, this component may use a mix of implicit and explicit ties between users. Two users may be deemed similar if they share the same interests, for example they use the same tags, they tag the same items, they live in nearby geographical addresses, or are friends real life. Based on user similarity values (computed using various measures such as Jaccard distance between users' actions, cosine in the case of tags, or simply, users having demographics attributes in common), the second task is the construction and maintenance of such networks in the *Network Construction* component. The result of this component is a user-centric network $\langle u, \{v\} \rangle$ composed of all users v similar to u (according to the various similarity functions). In del.icio.us for example, one option is to use a user's explicit network in order to recommend places to visit in a city. Finally, *Index and Cluster Generation* is another key component in search and recommendation applications. It is responsible for organizing data to optimize its further exploitation. This component identifies appropriate (structural or social) criteria for grouping users or items and appropriate mechanisms for ranking and selecting them. For example, when multiple networks and results are available, *Index and Cluster Generation* is used to determine which one is most relevant to a user based on current information needs.

In other research areas, data preparation has been identified as a cornerstone of application development. A data preparation step has been recognized in data mining (Pyle, 1999; Refaat, 2010) and includes *acquisition and integration*, *enhancement and enrichment*, and *transformation* and a library of tools implementing specific operations in SAS for data preparation were developed. No language or algebra as in SOCLE was provided.

2.2. Applications

We have examined a large number of social applications and identified four examples. These examples are far from being exhaustive but we believe they highlight with different complexities search and recommendation applications and analytics applications. The first example is a website recommendation application for individual users. The second is a movie recommendation application for adhoc user groups. The third application extracts sentiment from tweets for analytics purposes. The last one combines analytics and queries to generate travel itineraries from Flickr photos.

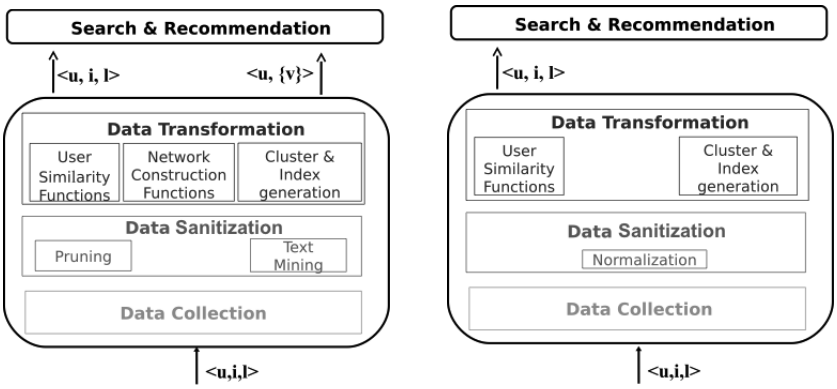
2.2.1. Website Recommendation in Delicious

Del.icio.us is a social bookmarking and tagging site. Users can subscribe to their friends' feeds in order to learn about their latest bookmarked URLs. They can also view hotlists (most popular URLs) and browse tags to find related URLs. Several efforts proposed a variety of methods for producing customized hotlists (as recommendations for users) and evaluated their effectiveness on del.icio.us datasets (Maniu, Cautis, 2012; Schenkel *et al*., 2008; Stoyanovich *et al*., 2008). A user specifies a query (e.g., list of keywords) and the search and recommendation applications return the top-k items (URLs) satisfying the query. If the query is empty, the application returns a personalized hotlist. If the query is a set of keywords, the application returns the highest scoring items in the user's network.

The del.icio.us dataset is a set of triples of $\langle u, i, l \rangle$ where l is a tag. Figure 2(a) instantiates the architecture of SOCLE for this application. The *Data Collection* component gathers data via the del.icio.us API and integrates collected data into this unique format. The *Pruning* component removes all URLs that were tagged by few users (often URLS tagged by less than 10 users are pruned) as well as uncommon tags (in most cases we examined, tags used by less than 5 users were pruned). Other options may include *Text Mining* to extract topics from tags (Amer-Yahia, Huang, Yu, 2009) The *User Similarity* component provides adequate functions to measure how close two users are in terms of friendship or common tagging interest for example. This step constitutes the holy grail of search and recommendation and has been explored extensively in the examined literature. Based on these functions the *Network Construction* component builds a user-centric network. This transformed data is used to construct quality hotlists. If the user specifies keywords, the *Index and Cluster Generation* provides per user and per keyword indices that can further be exploited (Amer-Yahia *et al*., 2008).

2.2.2. Group Recommendation in MovieLens

In many application scenarios, users get together to consume content of interest, e.g., a regular family gathering over the week end, or on an adhoc and ephemeral set of people attending a recital. Group recommendation is more challenging than individual user recommendation since it needs to consider not only individual needs but also disagreement among group members. In an individual user recommendation as seen previously with del.icio.us, recommendation applications build ranked lists of



(a) Delicious URL recommendation (b) MovieLens group recommendation

Figure 2. Instances of the SOCLE *architecture*

items. In group recommendation, the same item may have different predicted ratings for each group member and a consensus function (e.g., Least Misery or Aggregated Voting (Park *et al.*, 2008)) is used to aggregate individual user ratings into a group rating for each item (Roy *et al.*, 2010; Amer-Yahia, Roy *et al.*, 2009; Schafer *et al.*, 2007; Chen *et al.*, 2008; Park *et al.*, 2008)

Figure 2(b) instantiates the architecture of SOCLE for group recommendation. Data preparation starts by collecting known user ratings. That is the case for MovieLens whose rating information is available in a dump of ASCII files. *Data Collection* component wraps the collected data into $\langle u, i, l \rangle$ triples where l is the rating of u for i. The *Normalization* component processes ratings in order to make them comparable. A typical normalization formula recomputes each user's ratings in light of the scale used by a user. For example, to make the ratings of users u and v comparable, their ratings are transformed according to their respective lowest and highest ratings (Ricci, Shapira, 2011; Han *et al.*, 2004) The result is then used by the *Index and Cluster Generation* component to form user groups, based on group size and group cohesiveness (how similar/different users are in a group). Group size and cohesiveness are key factors in evaluating group recommendations. *Index and Cluster Generation* uses a similarity function provided by *User Similarity* in order to determine user groups. The similarity function takes into account user ratings to reflect how similar two users are in their movie preferences. The result of *Index and Cluster Generation* is a set of groups of varying size and cohesiveness that can then be used by different consensus functions to produce and evaluate group recommendations.

2.2.3. Social Analytics on Twitter

Twitter is a popular microblogging service. It allows users to publish and read short messages called tweets. The length of a tweet is restricted to 140 characters. Many social analytics applications apply their analytics algorithm on tweets especially for phrase and sentence level sentiment classification. Some analytics applications on

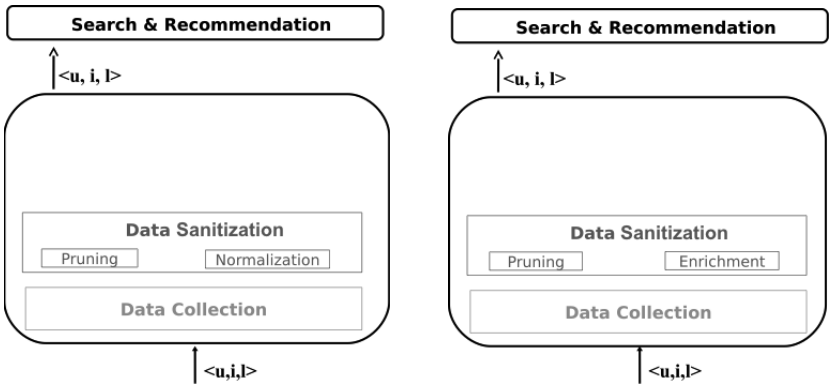
Twitter are sarcasm recognition over a collection of 5,9 million tweets (Davidov *et al.*, 2010), or automatic classification of sentiment (e.g., positive or negative) of Twitter messages (Go *et al.*, 2009).

Figure 3(a) instantiates the architecture of SOCLE for Twitter sentiment analytics. The data preparation phase for this application (and other similar ones (O Connor *et al.*, 2010; Paul, Dredze, 2011)) comprises two important tasks. Firstly, *Data Enrichment* such as substitution of some meta-tags with predefined tags (Davidov *et al.*, 2010) is usually performed. An example is the detection of URLs or other Twitter users in a tweet and their substitution with special tags such as [LINK] and [USER]. Another enrichment is mapping emoticons to two defined emoticons expressing positive and negative feelings, for instance the following emoticons :), :-), :) and :D are all mapped to :). Secondly, *Data Pruning* removes irrelevant tweets (Davidov *et al.*, 2010) or any tweet containing both positive and negative emoticons. That may happen if a tweet contains two subjects or is a re-tweet (Go *et al.*, 2009). The pre-processed data (pruned and enriched) is an input for various algorithms and/or classifiers implemented by the *Social Application Logic* layer. The authors of (Davidov *et al.*, 2010) identify in a semi-supervised way sarcastic tweets using pattern recognition and matching tools whereas those of (Go *et al.*, 2009) use machine-learning algorithms for classifying messages in Twitter as either positive or negative.

2.2.4. Itinerary Extraction from Flickr

Our last example is travel itinerary extraction from Flickr (De Choudhury *et al.*, 2010). Travel planning especially itineraries is a difficult and time consuming task. These applications use the experience of millions of travelers who share their itineraries via rich media data support such as photos. The application goal is to extract the itinerary of each traveler by mapping photos into Points Of Interest (POIs) and aggregate actions of many travelers into coherent queryable itineraries. Once those itineraries constructed, they can be queried to find itineraries from a point of interest to another satisfying certain time constraints specified by the users.

Figure 3(a) instantiates the architecture of SOCLE for itinerary extraction from Flickr. Before constructing itineraries, data is pre-processed. All photos taken by residents are filtered and the application only keeps photos related to touristic sightseeing. The photo timestamps are also used to prune photos with a time uploaded smaller than the picture time taking (this may be due to a dysfunction in the camera). Such tasks belong to the *Pruning* component. Then, the *Enrichment* component associates photos with their related POIs. POIs are obtained from various sources such as Yahoo! Travel and Lonely Planet. The association process can be geo-based or tag-based. The former relies on matching the photo geo location to the POI geo location in case these two information are available. The latter matches the photos' tags to the POIs names using well-known substring matching methods.



(a) Twitter sentiment extraction (b) Flickr itinerary extraction

Figure 3. Instances of the SOCLE *architecture*

3. Data and Storage Models

In this section, we identify the general requirements for SOCLE in terms of modeling BSD, we survey the related work in social and graph models, and we discuss the SOCLE data model and the main opportunities in designing it.

3.1. Requirements

There are two main requirements when modeling BSD: *that of designing an expressive enough model to represent BSD and enable their efficient manipulation and that of choosing the right storage backend for BSD to enable their efficient retrieval*. Evidently, the choice of model to use to represent BSD affects the data manipulation language and vice versa. Section 4 discusses the requirements of a data manipulation language for SOCLE and describes an algebra based on the model we will propose in this section.

The applications we consider in Section 2.2, use input data in the form of triples $\langle u, i, l\rangle$ to represent a user $u \in \mathcal{U}$ that assigned the label $l \in \mathcal{L}$ to an item $i \in \mathcal{I}$. Hence, the first requirement is to design a model able to represent all the information handled by common social applications. To do that, a label may be a tag as in del.icio.us or Flickr, a rating as in MovieLens, a review as in Yelp, a comment as on Al Jazeera, or sentiment as in Twitter. The natural choice for representing social networks is to use graphs where users and items are nodes, and relations are edges. For instance, the triple $\langle Sue, Titanic, 5\rangle$ which specifies that Sue rates the movie Titanic with the score 5 is mapped to two nodes Sue and $Titanic$, and an edge between them that represents the relation "rate". Nodes may have a variety of attributes, associated with users such as their demographics (age and location) or with items such as title and actors for a movie. Edges may correspond to explicit ties such as tagging between a user and an item or friendship between two users. Alternatively, they may correspond

to implicit ties, deduced through some further analysis such as similarity in tagging actions or people living in the same neighborhood.

Beyond representing raw and transformed social data, an appropriate model must enable efficient data manipulation. Hence, it is going to be used as a basis for defining algebraic operations that manipulate instances of that model. Therefore, it should be powerful enough to address the requirements of those operations.

Storing BSD is our second big challenge. Indeed, raw and transformed BSD have to be efficiently stored in order to enable undoing some data preparation steps efficiently. For example, storing computed user similarity networks would enable their partial recomputation when another similarity function is used.

It is also sometimes necessary to store nodes together, such as a user and his networks. To illustrate that, let us assume the example of John who would like to go to Australia, with a personalized itinerary, by visiting several intermediate countries. He currently lives in France. He would like to use connections in his social network, in order to find a simple path from his city to a city in Australia and have a roof to sleep at each step. John would like the path to satisfy a number of requirements: it should consist of at most five persons, the person at the end of the path should live in Australia, the guests along the path do not live in France, except for the first one, because John does not have a direct friend abroad. The output paths should be stored with the user John. The storage has to be optimized for data represented in a graph, with the possibility of storing nodes and edges. It also has to be optimized for searches exploiting data locality, from one or more root nodes.

In light of the above requirements, we propose to review the existing literature in modeling social data, namely SoQL (Ronen, Shmueli, 2009), SociQL (Serrano *et al.*, 2007), BiQL[2] (Dries *et al.*, 2009), SocialScope (Amer-Yahia, Lakshmanan, Yu, 2009), and SNDB[3] (Cohen *et al.*, 2013).

3.2. Related Work

All social networks databases have at their formal foundation variants of the basic mathematical definition of a graph. Nevertheless, the graph model of every system has a particular combination of features. For instance, it can be directed or undirected, with labeled or unlabeled edges and nodes, with attributes in edges and nodes, and with hyperedges. This yields a spectrum of more or less generic data models, that we now present.

Nodes and Edges At the basis of all graph models, nodes represent people and edges represent relationships among people. This allows to define, for instance, that *Alice is a friend of Ben*, by representing *Alice* and *Ben* with two nodes connected by an

2. This is a database developed for general networks, thus including social networks.

3. The acronym refers to the title of the article where the system is proposed: *A Social Network DataBase that learns how to answer queries.*

edge labeled with *isFriendOf*. However, in more general models nodes rather represent *resources* (or items), that are *physical* and *abstract* entities, like items and topics. This allows to represent *Alice tagged the movie Titanic*. Different kinds of nodes are allowed in SociQL, SocialScope, BiQL, and SNQL. Edges in the graph can be either directed or undirected. Notice that while *isFriendOf* is a symmetric relation, *tagged* is an asymmetric one. For symmetric relations, undirected edges suffice. However, because most of the interesting relations are asymmetric (e.g., *tagged*) directed edges are needed. Directed edges are supported by all systems, except for SoQL that employes the so-called reciprocal friends model (Ronen, Shmueli, 2009).

Attributes. Nodes and edges can have a set of attribute-value pairs to define richer content. Node attributes allow to have an enriched profile of a user or an item, by adding for instance an email address, a job profession, or geographic coordinates. Edge attributes allow to represent various types of relationships and interactions among people, by adding for instance a description of the edge, a weight representing the uncertainty of a relation, or a timestamp denoting when the relation took place. Node attributes are supported by all systems, while edge attribute are not supported by SoQL and SociQL.

Hyperedges. A social network graph can also have hyperedges. These are edges involving a *set* of nodes, that are used to model the notion of *group* of users and interactions. For example, an hyperedge can define the co-authorship of more than two persons (represented as nodes) for a particular book.[4] Conceptually, hyperedges allow to go beyond binary relations between resources of the social network graph, and are roughly equivalent to n-ary relations. BiQL, SNQL and SNDB graph data model are the most general ones, and support hyperedges.

3.3. Opportunities

Implementing a framework for BSD preparation in building social applications requires rethinking appropriate data models, and mainly how to store BSD in its different forms: raw, sanitized or transformed.

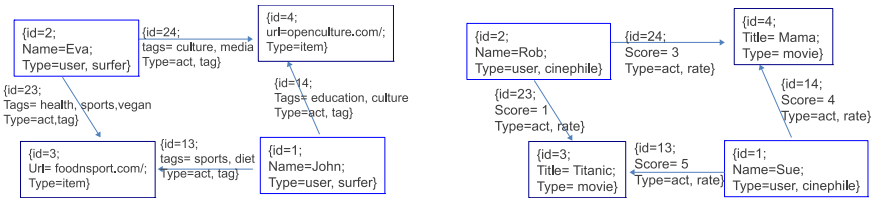
3.3.1. BSD Model

First and foremost, modeling BSD to enable efficient manipulation is a big challenge. We adopt a graph model for representing social content. Intuitively, nodes in the graph represent users and items, and links represent connections and activities between users such as friendship and between users and items such as tagging and rating actions. Each node or link has a unique id. The graph model described here is a logical model that is not tied to any specific physical implementation.

We adopt and extend the data model defined in (Amer-Yahia, Lakshmanan, Yu, 2009). That model is powerful enough to represent raw, sanitized and transformed

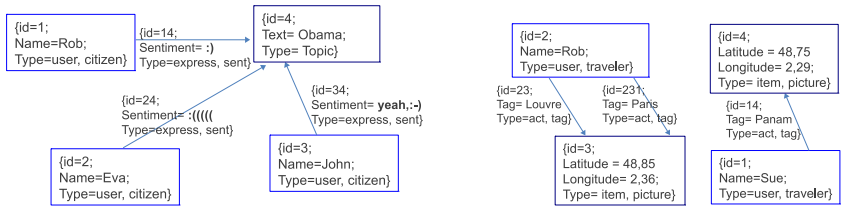
4. Attributes such as the name of the book are represented as attributes of the hyperedge

social data. Nodes and links contain structural attributes, including a mandatory type attribute. We adopt a flexible (i.e., schema-less) typing system and allow the type attribute to have multiple values. For example Fig. 4(a) represents two users, Eva and John, who tagged websites with different tags; in Fig. 4(b), Sue and Rob rate movies whereas Eva, John and Rob express their sentiments on Obama entity in Fig. 5(a). Finally, Fig. 5(b) represents users who tagged the same picture with different tags.



(a) Delicious data model (b) MovieLens data model

Figure 4. Examples of social data in the SOCLE *model*



(a) Twitter data model (b) Flickr data model

Figure 5. Examples of social data in the SOCLE *model*

Our typing system gives us the flexibility of creating new types through content analysis (e.g., if entities are extracted from text, their type can be added). We also maintain an evolving catalog of basic types, including user, item, topic, movie, connect (e.g., friend), act (e.g., tag, rate, review, click, visit). Those basic types are adequate for modeling most of the social content sites we have encountered.

To easily represent interactions, edges may be directed or undirected. Edge attributes can be user-defined or system-generated (e.g., computing the number of visited cities).

3.3.2. BSD Storage

The key challenge in data storage in general is to allow for its efficient retrieval. BSD storage has two additional challenges: first, data preparation (in particular, its data sanitization and data transformation steps) may generate new instances (in entity identification in text mining for example), new data types (in data enrichment for example), second, the same data instances may be accessed together or separately (for example, a user's network formed by other others should be accessed together to compute recommendations or separately whether its that one user's recommendations

that are computed or recommendations for users in his network). We here discuss the opportunities raised by such peculiarities.

Native vs non-native stores. There are numerous ways to encode and store a graph, some of which take advantage of natural index-free adjacency which is crucial for fast, efficient graph traversals (such as answering questions like "which cities did my friends visit that I did not visit yet?"). This means that each data element points directly to its inbound and outbound relationships, which in turn, point directly to related nodes, and so on. This ability is present in Neo4j (*Neo4j the Graph Database*, n.d.), a graph database useful for managing and querying highly connected data. in Neo4j, both nodes and relationships can hold properties in a key/value fashion.

In a non-native graph store (e.g., a relational database), there is a collection of tables of data items. For instance, to store data in Fig. 4(b), three tables (Fig. 6) can be used. The values of user (respectively movie and edge) attributes are given in table T_{users} (respectively T_{movies} and $T_{ratings}$).

$T_{users}(idu, name)$

1	Sue
2	Rob

$T_{movies}(idm, title)$

3	Titanic
4	Mama

$T_{ratings}(idu, idm, rate)$

1	3	5
1	4	4
2	3	1
2	4	3

Figure 6. Example of MovieLens data in a relational database

In a non-native graph store, indexes are used to link together nodes, whereas in a native graph store each node acts as a kind of micro-index of other nearby nodes (thereby inducing low retrieval costs). Conversely, in a native graph store, there is no requirement to maintain large global indexes to reconstruct subgraphs (at greater computational costs).

Hybrid stores. In practice, the best performance may be obtained from a hybrid storage (e.g., use of a relational database for indexing nodes and use of a native storage to benefit from index-free adjacency). This has been already found as an effective solution (Barbosa *et al.*, 2009) for storing and querying an XML corpus. The idea is that for instance, some XML documents have both very structured and very unstructured parts. Thus, one can use different storage models and systems to store different document parts. For instance, a sequence of tree nodes with the same tag, can be either stored in a relational table, or in a native XML storage system. Of course, this implies the development of different access methods corresponding to the different storage mappings and account for *potentially contradictory* data access requirements as discussed at the beginning of this section.

4. Languages and Algebras

In this section, we examine existing languages and algebras for querying and transforming social data and their applicability to the data preparation steps described in Section 2. We first discuss the requirements of a data manipulation language for SOCLE and review some of the related work. Then, we present the SOCLE algebra through examples inspired from our selected applications in Section 2.2. Finally, we summarize the main opportunities in developing *a data manipulation language* for BSD preparation.

4.1. Requirements

The SOCLE data manipulation language consists of a set of graph transformation primitives based on the following requirements.

Expressivity. Querying nodes, edges, and their attributes, in a social graph are fundamental operations, that however do not suffice for expressing data preparation operations when building social applications. The reason is that they find their applications mainly in *Data Sanitization*, for example in pruning operations. As we shall discuss, *Data Transformation* operations are ill-represented.

One essential operation the language must support is aggregate queries over sets of edges and nodes. Aggregates are fundamental for instance for value normalization in *Data Sanitization*, and when computing similarity networks for users in *Data Transformation*. Once aggregates are computed, their results are stored in new nodes and edges. Creating new edges and nodes is also helpful for restructuring a social network graph. For instance, the names of locations that are encoded as tag attributes of graph edges (see Figure 5(b)), may be needed in the form of graph nodes for itinerary extraction. Analogously, for website recommendation, one may want to represent as nodes the topics hidden in tags that reside on edges (see Figure 4(a)).

Restructuring a network is mainly done during *Data Transformation*. Top-k query answering is very important, for search and recommendation, to refine intermediate results and retain the most relevant ones. The language should also allow invoking external tools to perform operations such as mining and clustering. Those tools are used repeatedly to compute a measure of similarity and closeness of a set of nodes, perform node clustering, apply topic discovery on tags, or extract sentiment from text. They play a prominent role in *Data Transformation*. Analogously, it should be possible for the language to integrate external data sources (such as Lonely Planet to extract landmarks in a city), and mapping them to the SOCLE data model. This is needed to enrich data during *Data Sanitization*.

Declarativity. This is considered a crucial aspect of any data manipulation language. Declarative access to data permits fast deployment and easy maintenance of applications, in the spirit of logical and physical data independence. For building social applications, this is even more important since developers may have a strong

knowledge on a particular social application, but choosing the best implementations for data sanitization or data transformation will often be beyond their abilities.

Closure and invertibility. The graph manipulation operations expressed in SOCLE are required to be *closed* and *invertible*. Just like SQL or XQuery, the closure of the SOCLE language aims to ensure that each operation in the language takes as input one or more instances of and outputs an instance of the SOCLE model. Therefore, operations can be composed any number of times and in any order. Invertibility is unique property of the SOCLE language. It aims to guarantee that each sanitization or transformation operation admits an *inverse* allowing to undo and backtrack computation. This property is important to enable the evaluation of different application semantics, without rebuilding applications from scratch. For instance, one may want to change the threshold values of the user similarity functions or those used in an external clustering tool. Ensuring fine-grained invertibility (at the level of each operation in the language), will enable optimized execution plans where bulk pruning and aggregations, typical of *Data Sanitization*, are avoided.

Languages / Features	BiQL (Dries et al.,2009)	SocialScope (Amer-Yahia et al.,2009)	SoQL (Ronen et al.,2009)	SNQL (Martin et al.,2011)	SNDB* (Cohen et al.,2013)
Creation					
Link creation	✓	✓		✓	
Node creation					
Graph creation					
Hypergraph creation					
Graph intersect/union/diff					
Querying nodes					
Top-k nodes selection	✓	✓	✓		✓
Node selection					
Querying path					
Path selection	✓	✓	✓	✓	
Path matching					
Transitive closure					
Querying subgraphs					
Subgraph selection	✓	✓	✓		✓
Subgraph matching					
Advanced features					
Ranking	✓	✓	✓	✓	✓
Clustering					
Aggregation					

Figure 7. Features of social network languages

4.2. Related Work

Our related work into two subsections: social languages and algebras which are directly applicable to social data preparation, and other models for representing and processing data.

4.2.1. Social languages and algebras

Figure 7 summarizes data manipulation languages for social graphs and their features. It also provides a bibliographic reference for each language. These languages are tailored to social network analysis, and are used to query paths, create new networks, perform aggregations, and run data mining algorithms. We discuss some of their main features that are essential in our selected applications (Section 2.2).

Paths. A feature that makes social network databases akin to graph databases is the possibility of querying *paths*. A path is a sequence of nodes and edges in the graph that, for instance, represents the connection between two individuals in a social network. In social networks, it is important to perform path selection, path matching, and compute transitive closures. Path selection consists of retrieving all paths that satisfy some constraints. In its simplest form, a path selection can be expressed through a conjunctive query. This is expressible of course in any social network database. However, systems like SoQL and SociQL allow also to query paths according to i) a set of conditions on nodes and edges attributes and labels, ii) a distance from a fixed resource node. Path matching is a means of finding pairs of nodes that are connected by a path, whose sequence of edges and nodes matches some *regular expression* pattern. This is supported only by BiQL and SNQL. This kind of expressions have been extensively studied during the development of several graph-database systems (Consens, Mendelzon, 1990), and are described in comprehensive surveys (Wood, 2012; Angles, Gutiérrez, 2008). In a nutshell, the most important feature of such expressions is the Kleene-* operator, that allows to denote the transitive closures of relations.

Aggregation. Aggregation is critical for most analysis tasks on social content graphs. An aggregate is a function that applies to a set of objects yielding an aggregation result. Almost all languages we reviewed can express popular functions like summation, count, average, minimum and maximum (see Figure 7). The SocialScope algebra features two different types of aggregation: over nodes and over edges. SoQL supports also aggregation queries over paths. BiQL supports aggregation over paths and nested aggregate functions, that are needed for instance to get the maximum/maximum sum of weights for a set of paths satisfying a condition. For supporting aggregates, it is important to be able to create new nodes and edges where the aggregation results are stored. In BiQL and SNQL, one can radically change the social network graph, for instance by making edges and attributes become nodes. SocialScope supports graph composition, that is the possibility to create a graph induced by new edges, that are composed from edges belonging to two different source graphs.

External tools. To better analyze the data, some systems propose to also integrate a set of mining capabilities tailored to social network analysis. Ranking and top-k query answering are used in BiQL, SocialScope, and SNDBto reduce intermediate results. BiQL also supports calling external clustering tools.

4.2.2. Other models and algebras

We briefly review the closest non-social data representation and manipulation models and primitives in this section.

XML and Query. Another possible way of doing data preparation is by exploiting the universality of the XML format, and the XQuery and XSLT programming languages (Chamberlin *et al*., 2007; Kay, 2007). These languages are specifically designed to write (XML) data transformations, and can express the various data preparation tasks. XQuery is a functional language whose core, XPath, is also at the basis of the template-based language XSLT. These are however in contrast with our approach, where we directly define the algebraic operators to manipulate social graphs. Most importantly, the XML programming languages are designed and fully optimized to process tree-shaped data. Despite the fact that we can easily instantiate social graphs in the XML data-model, the joins needed to reconstruct graphs, stored and processed as a trees, become the bottleneck for computation.

Process algebras. Process algebras (Fokkink, 2007) model concurrent systems. They are used as a high level tool describing the interactions, communications and orchestrations of a set of processes. If we consider each module of the SOCLE architecture as a process, we can use process algebra languages (Fokkink, 2007) to describe the execution of each of module and represent the interaction between modules. For instance, a process algebra provides a set of operators such as parallel composition of processes or specification of which channels to use for sending and receiving data and process replication. But these languages are not suitable for describing data and data transformation. In SOCLE, manipulating and storing data is a crucial aspect that captures all the transformation data goes through in data preparation.

Workflows. Workflows provide a way of capturing a process so that results can be reproduced and the method can be reviewed, repeated and adapted. A workflow is usually a pipeline of different processes, described at a high level so that programmers do not need to be concerned with low-level programming. One major strength of workflows is their visual component allowing a large scale management process execution. Workflows have been used in many different areas (service oriented architecture, agent based model, etc) and many languages (Mathias Wesken, 2006) are dedicated to helping developers construct and execute their workflows. As powerful as it can be this precise description of a multi-step process to coordinate multiple tasks is far away of a declarative way. Social application developers will need to acquire a specific knowledge in data preparation processes, in order to describe and execute their workflows whereas an algebra will allow them to describe what they want to do and not how to do it.

4.3. Our algebra

We illustrate through examples the core operators of the SOCLE algebra for manipulating social graphs, allowing to perform selections, joins, and aggregates. It is inspired from the algebra of SocialScope (Amer-Yahia, Lakshmanan, Yu, 2009).

Selecting sub-networks of interest. Querying nodes, edges, and their attributes, is needed in all phases of *Data Sanitization* and *Data Transformation*, either to select a portion of the input social graph to analyze or in order to refine intermediary results. In particular, this is at the basis of all pruning tasks. We present two examples of selections made during website recommendation on del.icio.us.

"Which users are interested in sports sites?". This query can be written in the SOCLE algebra by means of two operators corresponding to the selection on nodes and links, that we denote by σ^N_C and σ^L_C respectively, where C is the filtering predicate.

$$\sigma^N_{type=user}(\ \sigma^L_{tag=sport}(\ G\ )\ )$$

The inner expression selects all edges with a tag *sport* in the graph G. These correspond to all tagging actions concerning sports sites, made by users. Then, the whole expression returns a (null) graph constituted of the set of nodes of type *user* associated to that edges. Consider the graph in Figure 4(a). The evaluation of this expression on the graph yields a social network graph constituted of the two nodes corresponding to the users *Eva* and *John* respectively.

"Which topics interest John?". This query is of crucial importance in website recommendation, since it feeds the successive research of all users that are interested in the same topics as *John*. Finally, it will be possible to suggest to *John* new topics that similar users already expressed, and he may like.

The query can be written with a selection of all outgoing edges of the node representing *John*. The outgoing edges of the node representing *John* are computed by using a semi-join operator between two graphs, that we denote by $G_1 \ltimes_\delta G_2$. The directional condition δ is a pair $\delta = (\delta_1, \delta_2)$ with, $\delta_i \in \{\mathtt{s}, \mathtt{t}\}$, which defines that the join condition between the edges in the two graphs is either between two source nodes $(\mathtt{s}, \mathtt{s})$, or between two target nodes $(\mathtt{t}, \mathtt{t})$, or between a target and a source node $(\mathtt{s}, \mathtt{t})$, $(\mathtt{t}, \mathtt{s})$, belonging to the first and second graphs respectively. This condition is called directional because it fixes the "direction" of joining edges. For all graph edge connecting n_1 with n_2, we say that n_1 is the source node and n_2 is the target node of the edge. A node with no edges is also considered a source node. The following expression joins all edges in the graph G with the source node representing *John*.

$$G\ \ltimes_{(\mathtt{s},\mathtt{s})}\ \sigma^N_{name=John}(G)$$

This yields a graph G' constituted of all outgoing edges of *John* node in G. Since all types of links are in the result, we apply a further link selection $\sigma^L_{type=tag}(G')$, as in the previous example, in order to retain just those representing a tagging action. Consider the graph in Figure 4(a). The evaluation of this expression on the graph yields a social network graph constituted of the edges connecting the *John* node with those representing the sites `foodandsport.com` and `openculture.com`, respectively.

Aggregation. Aggregation operators are at the basis of computing many different statistics on the data. Here we present an application in *Data Sanitization* for group

recommendation in MovieLens, where we computed the min (and analogously max) movie rating value of a user. This is needed for the Data Normalization phase, where in order to compare ratings of different users, these are first settled to the same range.

"What is the min movie rating given by Bob?". This query can be written by first selecting all rating actions of *Bob* with selections as illustrated before, and then by running an aggregate query on top of them. The aggregation operator for links is denoted by $\gamma^L_\nu(G)$. Here $\nu = (C, \texttt{att}, \mathcal{A})$ is a triple containing all aggregation parameters, namely an aggregation condition C, an attribute `att` where to store the aggregation result, and the aggregation operation $\mathcal{A} \in \{\texttt{Sum}, \texttt{Avg}, \texttt{Min}, \texttt{Max}, \dots\}$. Let G_{Bob} be the graph containing all outgoing edges from *Bob* node, the number of friends of *Bob* is computed by the following expression.

$$\gamma^L_{(rating, \texttt{min\_rating}, \texttt{Min})}(G_{Bob})$$

In this case, we have a boolean condition driving the aggregation just saying that the attribute target of aggregation is *rating*. Hence, in order to participate to the aggregation, an edge must have a *rating* attribute. However, for other types of aggregates (e.g., `Sum`) and applications (e.g., itinerary recommendation) we employ standard predicates such as $rating > 2$ or $type = travel$. Finally, the particularity of the aggregation operator is that it *modifies* the graph, in order to store the aggregation result. Each edge satisfying the aggregation condition (in this case featuring attribute tag $rating$) is here replaced with a fresh edge featuring the attribute `min_rating`, whose aggregation value is the number of edges satisfying the condition. Consider the graph in Figure 4(b). The evaluation of this expression on the graph yields a social network graph constituted of the edges connecting the *Bob* node with those representing the movies *Titanic* and *Mama*. Both edges feature a fresh id, and the only attribute $\texttt{min\_rating} = 1$.

4.4. Opportunities

Evaluating an expression in the SOCLE language poses new opportunities, due to the sheer volume of BSD and to the complexity of the language operators. We describe some classic optimization problems, such as query evaluation and result maintenance, and their applicability to building social applications.

Query evaluation. SOCLE operations are algebraic expressions over graphs. Each operation has an associated execution plan. Often, many equivalent execution plans for a single expression are possible, and choosing the best is crucial for efficiency. This challenge can be tackled by estimating the cost of each operation, rewriting one expression to an equivalent one, and by choosing access paths for evaluating subexpressions. The new opportunity is the ability to combine a native and a non-native storage of the social graph and leverage adjacency and ad-hoc indexes.

Full/Partial recomputation. A unique challenge of the SOCLE framework is that it should efficiently permit an *iterative* development of applications, where continuous changes in an application's semantics are made by developers which are not satisfied

with the results after evaluation. These continuous changes immediately raise an optimization issue, since it is inconvenient to restart the whole evaluation process from scratch every time there is a change. Instead, it could be more efficient to revert some operations, and perform partial restarts. Of course, this cannot always be the case and it is thus interesting to understand the tradeoff between a full recomputation and a partial one. To carry over this task, we envision the use of statistics to estimate the cost of different strategies, and define a notion of operation invertibility as an extension of work on provenance (Binnig *et al.*, 2007; Cheney *et al.*, 2009). Finally, notice that this is different from the classic view maintenance problem, that arises when changes on the *data* occur (as opposed to changes in *computation*).

Result maintenance. Efficiently maintaining intermediate results of computation, after changes on the data is a classic optimization problem. We plan to leverage that work to deal with general updates on the data, such as the integration of an *external* data set allowing for a more comprehensive analysis, or the suppression of a redundant graph.

Efficient evaluation on BSD. Efficiently evaluating SOCLE expressions over large social graphs demands to devise (i) proper indexes to quickly access graph nodes and edges according to their attributes, (ii) cost estimation functions for driving algebraic optimizations, and (iii) storage strategies that reflects the topology of the graph so as to minimize the cost of joins between adjacent nodes and edges. In particular, the evaluation of aggregates would largely benefit from effective indexing and storage techniques. All of these are central questions for the future development of the SOCLE platform.

5. Conclusion and Future Work

We presented SOCLE, a framework composed of an architecture, algebra and language for data preparation in social applications. We examined a large number of efforts in building two families of social applications, recommendation and analytics, that manipulate Big Social Data (BSD). We also reviewed the related work in modeling and querying graph data and social data in particular. We proposed a taxonomy of the operations covered by proposed algebras and languages and showed via examples inspired from the applications we examined, that there is a need for a new algebra and language for expressing BSD preparation.

Our immediate plan of action is the development of a concrete application in which we will explore the challenges raised in this paper. We conducted preliminary experimentation on storing data on graph database (Neo4j) and relational database (MySql). We developed a recommendation application for MovieLens users that implements the *Pruning*, *Normalization*, *User similarity Functions*, *Network Construction Functions* and *Cluster and Index Generation* modules. This application uses collaborative filtering to find for each user a list of recommended movies. User similarity for collaborative filtering was based on computing the Jaccard distance between rated movies.

Our first results are promising and show some differences in performance between the two storage solutions (relational and graph). However, we do not have substantial experimental results to report at this time as this would require many more experiments to make sure our results are satisfactory. We are currently running more experiments with a variety of other applications including tweets analysis.

The architecture of SOCLE is flexible and can be extended to cover other data preparation modules. The SOCLE algebra and language are based on SocialScope (Amer-Yahia, Lakshmanan, Yu, 2009) based on a data model with an extensible type system and that allows the definition of new primitives. Later, we plan to explore other social applications and identify BSD primitives not covered by the SOCLE algebra.

References

Abbar S., Amer-Yahia S., Indyk P., Mahabadi S. (2013). Real-time recommendation of diverse related articles. In D. Schwabe, V. A. F. Almeida, H. Glaser, R. A. Baeza-Yates, S. B. Moon (Eds.), *Www*, p. 1-12. International World Wide Web Conferences Steering Committee / ACM.

Amer-Yahia S., Anjum S., Ghenai A., Siddique A., Abbar S., Madden S. *et al.* (2012). Maqsa: a system for social analytics on news. In *Sigmod conference*, p. 653-656.

Amer-Yahia S., Benedikt M., Lakshmanan L. V. S., Stoyanovich J. (2008). Efficient network aware search in collaborative tagging sites. *PVLDB*, Vol. 1, No. 1, pp. 710-721.

Amer-Yahia S., Huang J., Yu C. (2009). Jelly: A language for building community-centric information exploration applications. In *Icde*, p. 1588-1594.

Amer-Yahia S., Lakshmanan L. V. S., Yu C. (2009). Socialscope: Enabling information discovery on social content sites. In *Cidr.*

Amer-Yahia S., Roy S. B., Chawla A., Das G., Yu C. (2009). Group recommendation: Semantics and efficiency. *PVLDB*, Vol. 2, No. 1, pp. 754-765.

Angles R., Gutiérrez C. (2008). Survey of graph database models. *ACM Comput. Surv.*, Vol. 40, No. 1.

Barbosa D., Bohanson P., Freire J., Kanne C.-C., Manolescu I., Vassalos V. *et al.* (2009). XML Storage.

Binnig C., Kossmann D., Lo E. (2007). Reverse query processing. In *Icde*, p. 506-515.

Chamberlin D., Florescu D., Boag S., Fernández M. F., Robie J., Siméon J. (2007). *XQuery 1.0: An XML query language*. W3C Recommendation.

Chen Y.-L., Cheng L.-C., Chuang C.-N. (2008). A group recommendation system with consideration of interactions among group members. *Expert systems with applications*, Vol. 34, No. 3, pp. 2082–2090.

Cheney J., Chiticariu L., Tan W.-C. (2009, April). Provenance in databases: Why, how, and where. *Found. Trends databases*, Vol. 1, No. 4, pp. 379–474. Retrieved from http://dx.doi.org/10.1561/1900000006

Cohen S., Ebel L., Kimelfeld B. (2013). A social network database that learns how to answer queries. *CIDR*.

Consens M. P., Mendelzon A. O. (1990). Graphlog: a visual formalism for real life recursion. In *Proceedings of the ninth acm sigact-sigmod-sigart symposium on principles of database systems*, pp. 404–416. New York, NY, USA, ACM. Retrieved from http://doi.acm.org/10.1145/298514.298591

Davidov D., Tsur O., Rappoport A. (2010). Enhanced sentiment learning using twitter hashtags and smileys. In *Coling (posters)*, p. 241-249.

De Choudhury M., Feldman M., Amer-Yahia S., Golbandi N., Lempel R., Yu C. (2010). Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st acm conference on hypertext and hypermedia*, pp. 35–44. New York, NY, USA, ACM. Retrieved from http://doi.acm.org.gate6.inist.fr/10.1145/1810617.1810626

Dries A., Nijssen S., De Raedt L. (2009). A query language for analyzing networks. In *Proceedings of the 18th acm conference on information and knowledge management*, pp. 485–494. New York, NY, USA, ACM. Retrieved from http://doi.acm.org/10.1145/1645953.1646016

Fokkink W. (2007). *Introduction to process algebra*. Springer-Verlag.

Go A., Bhayani R., Huang L. (2009). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, pp. 1–12.

Han P., Xie B., Yang F., Shen R. (2004). A scalable p2p recommender system based on distributed collaborative filtering. *Expert systems with applications*, Vol. 27, No. 2, pp. 203–210.

Kay M. (2007). *XSL Transformations (XSLT) Version 2.0*. W3C Recommendation.

Maniu S., Cautis B. (2012). Taagle: efficient, personalized search in collaborative tagging networks. In *Sigmod conference*, p. 661-664.

Mathias Wesken F. P., Gottfried Vossen. (2006). *Workflow and service composition languages*. Springer.

Neo4j the graph database. (n.d.). http://www.neo4j.org/. (Accessed: 2013-05-20)

O Connor B., Balasubramanyan R., Routledge B. R., Smith N. A. (2010). From tweets to polls: Linking text sentiment to public opinion time series. In *Proceedings of the international aaai conference on weblogs and social media*, pp. 122–129.

O'Connor M., Cosley D., Konstan J. A., Riedl J. (2001). Polylens: A recommender system for groups of user. In *Ecscw*, p. 199-218.

Park M.-H., Park H.-S., Cho S.-B. (2008). Restaurant recommendation for group of people in mobile environments using probabilistic multi-criteria decision making. In *Computer-human interaction*, pp. 114–122.

Paul M., Dredze M. (2011). You are what you tweet: Analyzing twitter for public health. In *Fifth international aaai conference on weblogs and social media (icwsm 2011).*

Pyle D. (1999). Data preparation for data mining. In, Vol. 1. Morgan Kaufmann.

Refaat M. (2010). Data preparation for data mining using sas. Morgan Kaufmann.

Ricci F., Shapira B. (2011). *Recommender systems handbook*. Springer.

Ronen R., Shmueli O. (2009). Soql: A language for querying and creating data in social networks. In *Icde*, p. 1595-1602.

Roy S. B., Amer-Yahia S., Chawla A., Das G., Yu C. (2010). Space efficiency in group recommendation. *VLDB J.*, Vol. 19, No. 6, pp. 877-900.

Schafer J. B., Frankowski D., Herlocker J., Sen S. (2007). Collaborative filtering recommender systems. In *The adaptive web*, pp. 291–324. Springer.

Schenkel R., Crecelius T., Kacimi M., Michel S., Neumann T., Parreira J. X. *et al*. (2008). Efficient top-k querying over social-tagging networks. In *Proceedings of the 31st annual international acm sigir conference on research and development in information retrieval*, pp. 523–530. New York, NY, USA, ACM. Retrieved from http://doi.acm.org/10.1145/1390334.1390424

Serrano D., Stroulia E., Barbosa D., Guana V. (2007). Sociql: A query language for the socialweb. In *Icdt*, p. 269-283.

Stoyanovich J., Amer-Yahia S., Marlow C., Yu C. (2008). Leveraging tagging to model user interests in del.icio.us. In *Aaai spring symposium: Social information processing*, p. 104-109.

Wood P. T. (2012, April). Query languages for graph databases. *SIGMOD Rec.*, Vol. 41, No. 1, pp. 50–60. Retrieved from http://doi.acm.org/10.1145/2206869.2206879