Partitioning and Ranking Tagged Data Sources

Milad Eftekhar
Department of Computer Science
University of Toronto
milad@cs.toronto.edu

Nick Koudas
Department of Computer Science
University of Toronto
koudas@cs.toronto.edu

ABSTRACT

Online types of expression in the form of social networks, micro-blogging, blogs and rich content sharing platforms have proliferated in the last few years. Such proliferation contributed to the vast explosion in online data sharing we are experiencing today.

One unique aspect of online data sharing is tags manually inserted by content generators to facilitate content description and discovery (e.g., hashtags in tweets). In this paper we focus on these tags and we study and propose algorithms that make use of tags in order to automatically organize and categorize this vast collection of socially contributed and tagged information. In particular, we take a holistic approach in organizing such tags and we propose algorithms to partition as well as rank this information collection. Our partitioning algorithms aim to segment the entire collection of tags (and the associated content) into a specified number of partitions for specific problem constraints. In contrast our ranking algorithms aim to identify few partitions fast, for suitably defined ranking functions.

We present a detailed experimental study utilizing the full twitter firehose (set of all tweets in the Twitter service) that attests to the practical utility and effectiveness of our overall approach. We also present a detailed qualitative study of our results.

Keywords

Social media, Tagged data, Partitioning, Twitter

1. INTRODUCTION

In recent years, the explosion of data available through online communities and social networks (such as facebook, google plus, twitter) created a pressing need for their management and analysis. Extracting valuable information buried in such massive data is a real problem of vast significance in diverse industries (marketing, advertising, market research, finance, public relations, journalism to name a few). Millions of individuals around the world update their statuses, post millions of tweets, share links and interact with each other exchanging messages, images, videos etc. The numbers are staggering: over 330M tweets on a daily basis on twitter, 3.5B pieces of content shared on facebook every week and in excess of 130M blog posts generated daily.

Given such volumes, it is imperative to impose some form of categorization or grouping of this information as a first step towards understanding and consuming it. Understanding trending topics, discussions or identifying groups of related content is a major challenge. In this paper we utilize one unique aspect of such socially contributed information namely manual tagging inserted by humans. For example, millions of tweets on a daily basis contain human inserted tags (commonly known as hashtags) that aim to best describe the content of the tweet in order to facilitate discovery. This makes online discussion and referencing easier. In a similar fashion, popular social networks such as google+ adopt such human assisted tagging. This concept has also been prevalent from the early days of the blogosphere. In all cases humans insert one or more tags when they generate a piece of content; such tags are available along with the content, commonly precisely describing it in a concise manner or attributing it to a certain set of thematic or topical entities. As an example, a tweet about the Oscars event (e.g., "I am confident the Descendants will win an award tonight!") can be tagged via a hashtag "`#oscars`" or "`#oscars #nomination`" or "`#oscars #award`". Although there are no standards in the generation of tags, they are commonly highly descriptive of the content. Given the similarity in the use of tags in various services, we will use the terms hashtag and tag interchangeably in the remainder of this paper. Also to focus our presentation, we will carry our discussion using the popular micro-blogging service Twitter as an example. The same concepts and techniques carry over in all services utilizing human or machine inserted tags.

We present algorithms that operate on tags in order to group pieces of related content together. In particular, given the set of all tweets (messages generated on Twitter) for a specific time period (say a day), we develop algorithms to partition such tweets into meaningful groups utilizing the tags of the tweets. In addition, we develop notions of importance in each group so that instead of identifying all groups, only a selected subset of those can be identified fast. Given the volume of information involved (330M tweets on disk is over 500 GB), as a design principle we are seeking fast algorithms to conduct these operations.

Simple approaches such as ranking individuals or sets of tags by their frequency in the entire collection of tweets will

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 30th 2013, Riva del Garda, Trento, Italy.
Proceedings of the VLDB Endowment, Vol. 6, No. 4
Copyright 2013 VLDB Endowment 2150-8097/13/02... $ 10.00.

fail to adequately provide a solution. In the former case, a lot of context is lost; the latter case introduces the need for principled approaches to determine when it is beneficial to merge collections of tags into a single representation which is the by-product of our approach. Perhaps, the first solution that comes to mind is to model tweets and hashtags as a bipartite graph $G = (V, U, E)$ where V is the set of tweets, U is the set of tags and $(i, j) \in E$ if tweet i contains hashtag j. Utilizing this graph model, we can employ several graph partitioning or clustering techniques (e.g., biclustering techniques [3, 4, 7]) to create groups of associated tweets and hashtags. The most crucial problem with this approach (and the bipartite graph approach in general) is that it assumes that hashtags are independent, something that does not hold in reality.

In response to the raised concerns, we model the set of tweets using a lattice. Such a lattice aims to maintain tag correlations and at the same time provide a natural grouping of sets of related tags (Section 2). In this paper, we introduce and study two sets of problems in this lattice: (1) partitioning problems (Section 3) that segment the entire lattice into a number of partitions based on problem specific constraints and (2) ranking problems that produce only a specific number (specified as input) of "top" partitions for suitably defined measures of partition importance (Section 4).

We discuss two types of partitioning problems: the first aims to create the maximum number of partitions, while each one contains at least c tweets (for some constant c). The size constraint (at least c tweets in each partition) is introduced to ensure that we do not create small partitions; the objective (maximize the number of partitions), on the other hand, prevents from merging unrelated tweets into one single partition. We show that this problem is NP-complete and propose a 2-approximation algorithm (algorithm AMSP) to approximate its solution (Section 3.1). The second problem (Problem 2) aims to create k partitions such that the minimum size of a partition is maximized. In a sense, even though we fix the desired number of partitions, we aim to produce partitions balanced in size. Section 3.2 shows that this problem is NP-complete and present a heuristic approach (algorithm MMP) to address it.

The second class of problems we introduce (Section 4) aims to extract the "best" k partitions from the lattice. To quantify the goodness of a partition, we introduce a weight function. We refine our study by introducing two problems in this case as well: the first (Problem 3) aims to identify k partitions that in aggregate produce the maximum weight among all possible combinations of k partitions; we present an optimal linear algorithm (algorithm MST) to solve this problem. The second (Problem 4) identifies k partitions with the highest weight values. We show that this problem is NP-complete (Section 4.2) and present an approximation algorithm (algorithm AMWT, Section 4.2.1) as well as a faster heuristic algorithm (algorithm HMWT, Section 4.2.2).

To evaluate our developments, we utilize the Twitter fire hose, namely the set of all tweets flowing through the Twitter service. In Section 5, we evaluated our algorithms at full scale for days worth of twitter data (not samples of it). This is the first study we are aware of utilizing the entire Twitter fire hose and stresses the practical utility of our developments as evident in our experimental section. All of our algorithms can operate on the days worth of full fire hose, in minutes for all reasonable instances in the worst case, and in seconds typically. In addition we deploy a qualitative study demonstrating the goodness of our findings and compared our partitioning algorithms with some baseline techniques (Section 5.2). A literature review is provided in Section 6, followed by Section 7 concluding our discussion.

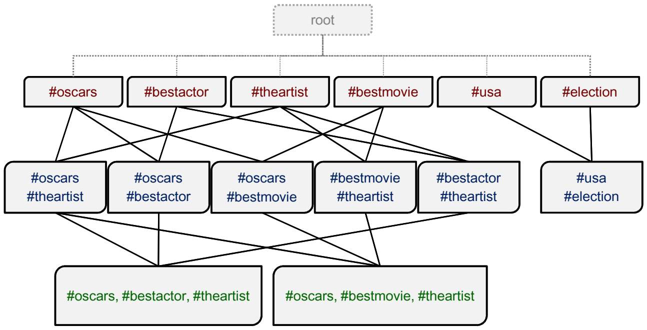


Figure 1: A sample lattice.

2. MODELING

Let $T = (t_1, t_2, \cdots, t_n)$ be a collection of tweets (n tweets), $H = \{h_1, h_2, \cdots, h_m\}$ be the set of **hashtags** (m hashtags), and $S = (s_1, s_2, \cdots, s_n)$ be the collection of hashtag sets (**tagsets**) corresponding to tweets. A tagset s_i is the set of all hashtags appearing in tweet t_i. For instance, the tagset associated to tweet "`#usa #election #obama` visits `#chicago` campaign headquarters" is {`#usa`, `#election`, `#obama`, `#chicago`}. Moreover, let d denote the maximum number of tags in tweets; i.e., $d = \max_{i:1 \leq i \leq n} |s_i|$. We model this tweet dataset by a lattice $L = (V, E)$ s.t.:

- There is a unique vertex in V corresponding to each non-empty tagset $s \in S$ and each subset of s.

  $$\forall s \in S \ \forall s' \subseteq s, \ \exists! v \in V : \ TS(v) = s'$$

  We associate each vertex v with a tagset $TS(v)$ and a size $N(v)$ where $N(v)$ identifies the number of tweets in the dataset having $TS(v)$ as their tagsets (the exact tagset not a superset/subset of it).

- There is an edge between two vertices $u, v \in V$ if and only if the tagset of one is a subset of the other and the difference between the length of their tagsets is 1. The length of a tagset is the number of hashtags in it. For instance, the length of the tagset (`#usa#nyc`) is 2.

  $$(u, v) \in E \Leftrightarrow TS(u) \subset TS(v) \wedge (|TS(v)| - |TS(u)| = 1)$$

Consider the hashtags of three tweets: $s_1 =$ (`#oscars #bestactor#theartist`), $s_2 =$ (`#usa#election`), and $s_3 =$ (`#oscars#bestmovie#theartist`). Figure 1 depicts the corresponding lattice.

The lattice framework aims to capture a natural property of the tagging process: humans use multiple tags to describe more specifically the content they produce (the tweet). In general, a tag set s_1 (e.g., `#usa #politics #democrats`) aims to describe more specific content than a tag set s_2 (e.g., `#usa #politics`) if $s_2 \subset s_1$. The lattice of Figure 1 lends itself to a very natural interpretation. Looking from top to bottom, higher level descriptions (as expressed by humans providing the tags) are at the top, becoming more specific as we follow paths emanating from the node. Moreover siblings are labeled with tags corresponding to related descriptions. Thus, such a lattice by construction has the property to group together related descriptions (as siblings) as well as impose ancestor-descendant relationships among general and more specific descriptions.

As the first phase to create the lattice, we perform some preprocessing steps to clean data. On Twitter, there are some hashtags that appear very frequently almost everyday such as `#oomf` (one of my followers), `#np` (now playing), `#nw`(now watching), `#rt` (retweet), `#nf` (now following), `#tfb` (teamfollowback), `#yolo`(you only live once), `#aries`, `#capricorn`, and `#libra` (the cardinal signs) as well as a few others. We consider those stop words and filter them. Moreover, we apply stemming techniques to merge hashtags with the same stem (e.g., `#oscars` and `#oscar`).

For each arriving tweet t_i, we need to update the lattice by adding vertices corresponding to the t_i's tagset s_i and all of its subsets $s' \subset s_i$. The number of these new vertices is at most $2^d - 1$. By utilizing proper hash functions, the addition and search processes can be done in $O(1)$. If the number of distinct tagsets is l, and each tagset contains at most d tags, the lattice contains at most $(2^d - 1) \times l$ vertices. On average, many subsets are shared between tagsets hence the size of the lattice is much smaller in practice. In real datasets, d is small. For instance, among 300 million tweets of Feb 26, there is essentially no legitimate tweet with more than 10 hashtags. Thus, the size of this lattice is easily manageable. In this paper, we assume d is a constant.

3. PARTITIONING

We utilize user-generated hashtags to partition the lattice generated from a collection of tweets. A **partitioning** of a lattice is a division of the lattice nodes into several connected disjoint sublattices such that the union of the vertices in these sublattices generates the vertex set of the original lattice. We call each sublattice, a partition and represent each partition by a **tree** (one of its spanning trees). Moreover, we refer to the **size** of a partition (tree) as the sum of its nodes' sizes; i.e., for a partition p: $|p| = \sum_{u \in p} N(u)$, where $|p|$ denotes the size of p and u is a node in p.

Generally in a partitioning problem, it is desirable to create partitions having suitable sizes. In extreme undesirable cases, one may assign each tweet/tagset to a separate partition resulting in many small partitions; or may assign all tweets/tagsets to one single partition resulting in spurious merging of irrelevant tweets. Intuitively, a good partition is the one that satisfies the following properties.

Property 1. ***(Size)*** *It is desirable to create partitions that are not small in size.*

Property 2. ***(Less number of nodes)*** *It is desirable to create partitions with a small number of nodes (distinct tagsets). Existence of many nodes in a reported partition means that tweets with different tagsets are placed together. Thus, that partition may group tweets about different events and this can lead to poor partitions. Since we represent each partition with a spanning tree, the number of nodes in a partition is related to the number of edges in its spanning tree and is conversely related to the total number of partitions.*

We introduce two complementary partitioning problems that attempt to satisfy these properties by fixing one property and trying to optimize the other one. The first problem is referred to as **MinSize** Partitioning; for a given minimum partition size, it aims to maximize the number of qualified partitions. Thus, given an a priori expectation of how many tweets (at least) should belong to a partition, we seek to maximize the number of all qualified partitions. In the latter, the number of partitions is provided and the goal is to create partitions with suitable sizes, namely the size of the lightest partition is maximized. We refer to this problem later as **MaxMin** Partitioning.

In Section 3.1, we formally define the MinSize Partitioning problem and show that it is NP-complete. Moreover, we present a 2-approximation algorithm to approximate its optimal solution. Section 3.2, formally defines the MaxMin Partitioning problem, demonstrates that this problem is NP-complete as well, and presents a heuristic solution.

3.1 MinSize Partitioning (MSP)

We aim to decompose the lattice into a number of partitions (represented by spanning trees) of size at least c such that the number of partitions is maximized (or equivalently the sum of the number of edges in all representing trees is minimized). Recall that the size of a tree is the sum of the sizes of its nodes. Since the only parameters that are important in a partition are the size of the partition and the number of edges in its spanning trees (equivalently the number of nodes in that partition), the choice of the spanning tree to represent a partition does not make a difference in this paper's problem definitions and algorithms.

Problem 1. *Let c be an (input) integer. Partition the given lattice $L = (V, E)$ to an unknown number of trees $\top_1, \top_2, \cdots, \top_r$ where $\top_i = (V_i, E_i)$, such that:*
1. $\bigcup_{1 \le i \le r} V_i = V$,
2. $\forall i, j : 1 \le i < j \le r; V_i \bigcap V_j = \emptyset$,
3. $\forall i : 1 \le i \le r; |\top_i| \ge c \; (|\top_i| = \sum_{u \in \top_i} N(u))$,
4. $\sum_{1 \le i \le r} |E_i|$ is minimized (equivalently r is maximized).

Theorem 1. *Minimizing the total number of edges in trees is equivalent to maximizing the number of created trees* [1].

Theorem 2. *Problem 1 is NP-complete.*

We propose an algorithm to approximate Problem 1 that generates high quality trees. An approximation bound for this algorithm is provided in Theorem 3. The main intuition is to separate lattice nodes (tagsets) with size larger than c from the rest of the lattice and consider them as potential partitions. In the remaining lattice, we randomly merge the nodes to create bigger partitions and stop when all partitions created contain at least c tweets. The pseudo code is provided in Algorithm 1.

Theorem 3. *The approximation bound of the solution discovered by Algorithm 1 is* 2.

Theorem 4. *The run time of Algorithm 1 is $\Theta(l)$ where l is the number of distinct tagsets.*

3.2 MaxMin Partitioning (MMP)

A variation of Problem 1 is the situation where one wants to decompose the lattice into a specified number of large partitions. In this scenario, the number of required partitions is given (say q) and we aim to segment the lattice into q partitions such that each partition is as large as possible. In other words, we intend to create q partitions such that the minimum size is maximized.

[1] Proofs are omitted due to space limitations. Please See the technical report [5] for complete proofs.

Algorithm 1: Approximation MSP (AMSP)

```
Build a new lattice $L'$ by removing all nodes with a size not smaller than $c$ and their adjacent edges;
Make a single-node output tree for each node in $L - L'$;
Identify the connected components of $L'$ and determine a spanning tree for each component;
Create a single-node output tree for each vertex in $L'$;
while There exists at least one unexamined edge in the spanning trees of the connected components of $L'$ AND an output tree with a weight smaller than $k$ do
    Choose an unexamined edge $e$, and merge output trees at both sides of $e$ if at least one of them has size smaller than $c$;
end
foreach output tree $\top$ with a weight smaller than $c$ do
    find and insert an edge between $\top$ and an output tree of $L - L'$;
end
return output trees;
```

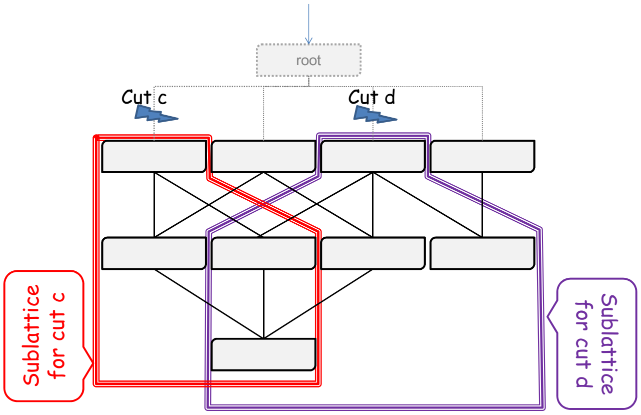


Figure 2: Cuts and sublattices.

Problem 2. *Let q be an input integer and $L = (V, E)$ be a lattice containing r $(1 \leq r \leq q)$ connected components. Segment L into q partitions $\top_1, \cdots, \top_q$, delineated by spanning trees, where $\top_i = (V_i, E_i)$ st.*

- $\bigcup_{1 \leq i \leq q} V_i = V$,
- $\forall i, j : 1 \leq i < j \leq q; V_i \bigcap V_j = \emptyset$,
- $\min_{i \in \{1,\cdots,q\}} |\top_i|$ *is maximized* $(|\top_i| = \sum_{u \in \top_i} N(u))$.

Theorem 5. *Problem 2 is NP-complete.*

We propose a generalized algorithm to address Problem 2 on a **lattice** (Algorithm 2). Our goal is to select $q - 1$ edges (cuts) inside the lattice that create a max-min q-partitioning. At the end of the algorithm, the partitions, represented by spanning trees, would be the sublattices under the selected cut edges. Figure 2 depicts sublattices corresponding to two sample cuts. The most critical concern with these sublattices is that they are not disjoint. To avoid this inconsistency, we define disjoint run-time sublattices (RSL) for cuts. The **sublattice** of a cut ζ ($SL(\zeta)$) is the set of all lattice vertices that are descendant of ζ, while the **run-time sublattice** of ζ ($RSL(\zeta)$) is the set of all descendant vertices that are not a current member of any other cut's run-time sublattice. We define the **size of a cut** as the size of its run-time sublattice (the sum of the sizes of all nodes in that RSL). The pseudo code for modifying RSL sets is illustrated in Function `RSL_Updater` in Algorithm 2.

Algorithm 2: MaxMin Partitioning (MMP)

```
   input  : lattice $L$, int $q$
   output: $q$ partitions
   // Initialization:
1  if $|children(root(L))| > 1$ then
2      Insert a new single edge on top of the root;
3  end
4  Assign all $q$ cuts to the single edge adjacent to the root;
5  foreach cut $c$ do
6      $RSL(c) = Null$; // RSL: run-time sublattice
7  end
8  For the first cut $f$: $RSL(f) = L$;
   // Iterations:
9  Calculate $S_{min}$ (the size of the current lightest cut);
10 Identify a move of a cut $c$ (downward to a neighboring edge with no cuts) leading to a run-time sublattice with maximum size ($S_c$);
11 if $S_c \geq S_{min}$ then
12     Execute the identified movement of cut $c$;
13     Execute Function RSL_updater for cut $c$;
14     Go to 9;
15 end

   // Function Definition:
16 Function RSL_updater:
17 Update $RSL(c) = RSL\_old(c) \bigcap SL(c)$; // RS_old(c) is the RSL of cut $c$ prior to its movement
18 Update $size(c) = \sum_{u \in RSL(c)} N(u)$;
19 Let $c'$ be the lowest predecessor cut of $c$;
20 Update $RSL(c') = RSL(c') \bigcup \{RSL\_old(c) - RSL(c)\}$;
21 Update $size(c') = \sum_{u \in RSL(c')} N(u)$;
```

Theorem 6. *The run time of Algorithm 2 is $\Theta(ql)$ where l is the number of lattice nodes.*

Example 1. Consider the lattice in Figure 3(a). We aim to divide this lattice into $q = 3$ partitions utilizing Algorithm 2. The first step is to insert an extra edge on top of the root and assign 3 cuts to it (Figure 3(b)). Initially, all vertices are assigned to the first cut, A. Thus: $S_A = 20$, and $S_B = S_C = 0$. Hence, $S_{min} = 0$. In the first step, there are 3 options (3 available edges downward) to move cut A with new sizes of $13, 12, 9$. We move this cut to the leftmost edge (size: 13). Therefore, four vertices remain in partition A and the rest are assigned to its first predecessor cut, B. Thus, $S_A = 13, S_B = 7$, and $S_C = 0$. Hence $S_{min} = 0$ (Figure 3(c)). In the next step, 4 options exist to move cuts. Cut A can move downward to two edges with new sizes 8 or 4 and cut B can move to 2 edges with new sizes 4 or 5. The max value is 8 (for cut A) that is greater than the current S_{min}. Thus, we move cut A downward to the left edge (size: 8) and update run-time sublattices and sizes $S_A = 8, S_B = 12, S_C = 0$. Thus, $S_{min} = 0$ (Figure 3(d)). In the next step, cut B is moved to the rightmost edge with size 6 (Figure 3(e)). Therefore, $S_A = 8, S_B = 6, S_C = 6$, and $S_{min} = 6$. The last options are moving cut A (size: 3), moving cut B (size: 1 or 2), and moving cut C (size: 4 or 2). The maximum value (4) is less than S_{min} (6). Thus, the algorithm terminates with a minimum size of 6 (Figure 3(f)).

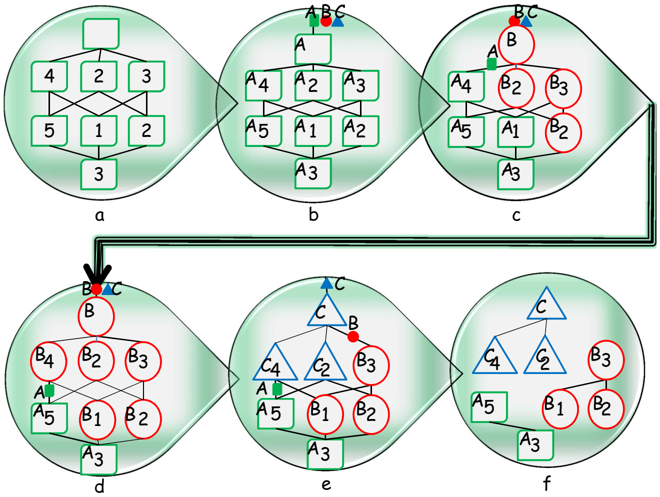


Figure 3: Running MMP on the lattice in (a).

4. TOP PARTITIONS

In this section, we consider ranking problems on the lattice. We define notions of "important" sets of nodes on the lattice and aim to quickly identify a number (provided as input) of such partitions (represented by their spanning trees). We define importance utilizing a weight measure. Using such weight functions, we address two problems with different objectives. Section 4.1 concentrates on identifying k partitions with the maximum sum of weights (**MaxSum** Top Partitions) and presents a linear algorithm to extract the optimal partitions. In Section 4.2, we aim to find k partitions such that the minimum weight of them is maximized (**MaxWeighted** Top Partitions). We formally define the problem and show that it is NP-complete. In Section 4.2.1, we propose an algorithm to approximate the optimal solution of this problem. A faster heuristic algorithm providing results of similar quality is provided in Section 4.2.2.

We start by explaining desired properties in the quality of partitions belonging to the final answer and then propose a weight function to capture them. As mentioned in Section 3, it is desirable to have large partitions (Property 1) containing small number of lattice nodes (Property 2). Due to the nature of ranking problems (not covering all tweets and searching for very important ones), the mentioned properties are necessary but not sufficient. Hence, we explain two extra desired properties for the ranking problems.

PROPERTY 3. ***(Maximizing coverage)*** *The top partitions should maximize the number of tweets they encompass; this is a natural requirement in order to report as many underlying events as possible. Thus, maximizing the fraction of tweets that are covered by the reported partitions is important.*

PROPERTY 4. ***(Complete importance)*** *The goals of maximizing coverage and minimizing the number of lattice nodes seems conflicting at first glance. When we add a node to a partition, in one hand, we gain since tweet coverage increases but on the other hand, we incur a penalty by increasing the number of nodes. We call a partition "completely important" if the gain we achieve by adding each single node, is bigger than the penalty we incur.*

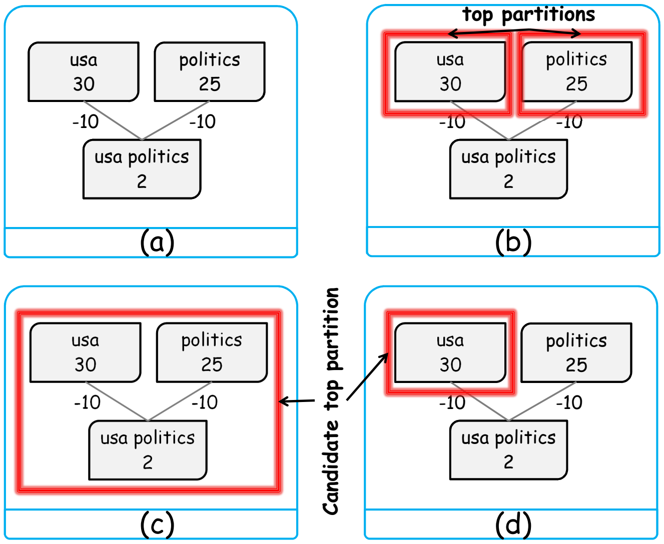


Figure 4: Complete importance property: (a) a sample sub-lattice (b) top-2 partitions (c)-(d) candidate single top partition

Assume that for each edge $e = (u, v)$ belonging to a partition p, $S_e(u)$ denotes the largest connected component of p containing u but not v and e. Formally, a partition p is completely important if for each edge $e = (u, v) \in p$:

1. $weight(S_e(v) \bigcup \{u\}) - weight(S_e(v)) > 0$
2. $weight(S_e(u) \bigcup \{v\}) - weight(S_e(u)) > 0$

To illustrate why the complete importance property is desired, we provide the following example. Consider the sub-lattice in Figure 4(a). It contains 3 distinct tagsets (`#usa`), (`#politics`), and (`#usa`, `#politics`). If we use sum of the partition weights to measure the goodness of the results, the best way to choose top-2 partitions for the lattice, as shown in Figure 4(b), is to choose nodes `#usa` and `#politics` with a total weight of 30 + 25 = 55.

If we want to select one top partition, however, the story is different. Figure 4(c) shows a candidate top partition. Although this partition has the largest weight among all possible partitions in the lattice, it is not a proper choice for us since it may group irrelevant tweets together. In this example, the large size of `#usa` (N(`#usa`) = 30) means that there is some attention to events related to `#usa`. The same happens for `#politics`. But by the small size of (`#usa`, `#politics`), we infer that there is less attention to politics in USA. In fact, there are many tweets about USA but not related to Politics (e.g. tweets about Oscars) and there are many tweets related to Politics but not related to USA (e.g. tweets about the Arab spring). Therefore, placing all these tweets together creates a group of heterogeneous tweets. In this situation, the candidate partition in Figure 4(d) that groups all tweets about `#usa` together is preferred. However, if the size of (`#usa`,`#politics`) is big enough, we can merge all 3 nodes in a group since there are events about USA politics that can play a connector role between USA tweets and politics tweets.

To render things concrete, in this paper, we utilize the weight function defined next, taking into account the properties established, to rank partitions.

Definition 1. (Weight function): Let $\alpha \in [0,1]$ determine the importance of the coverage vs. the number of nodes, V (V_p) be the set of vertices in the lattice (partition p), n be the number of tweets, u be a node in partition p, and $N(u)$ be the size of u. We define:

$$weight(p) = \alpha \times \sum_{u \in p} \frac{N(u)}{n} - (1-\alpha) \times \frac{|V_p|}{|V|}$$

The weight function is a weighted average of the fraction of coverage (Properties 1,3) and the negative of the fraction of number of nodes in each partition (Property 2). Thus, it aims to fulfill all three properties simultaneously. Moreover, this weight function sets up a tool to establish Property 4 (Lemma 1).

Lemma 1. *We define the weight of an edge $e = (u,v)$ as $weight(e) = \min(weight(\{u,v\}) - weight(\{u\})\ ,\ weight(\{u,v\}) - weight(\{v\}))$. Here $\{u\}$ is a partition with a single member u and $\{u,v\}$ is a partition with two members u and v. A partition p is completely important iff for each edge $e \in p$: $weight(e) > 0$. If the weight function is defined according to Definition 1, the edges weights can be simplified as $weight(e) = \alpha \min(\frac{N(u)}{n}, \frac{N(v)}{n}) - \frac{1-\alpha}{|V|}$.*

We define two complementary ranking problems attempting to satisfy the mentioned properties by fixing Property 4 and trying to optimize Properties 1,2,3 (by means of maximizing the weight function) in two possible directions. The first direction maximizes sum of the weight values of top-k partitions; the second one maximizes the minimum weight among top-k partitions.

We stress, however, that the algorithms proposed in this section do not exclusively operate using the weight function of Definition 1. Any function satisfying the general Property 5 can be utilized by our algorithms.

Property 5. *$\forall u \in V$: $sign(weight(p \bigcup \{u\}) - weight(p))$ is independent of p where p is a neighboring partition of u and $sign(x)$ determines if x is positive, negative, or zero.*

Property 5 states that the sign of the marginal gain, achieved by adding a node to a partition, depends on that node not the partition. In other words, we will gain, by adding node u to a neighboring partition p, iff we gain, by adding u to any other neighboring partition.

Example 2. The following weight functions satisfy Property 5.

- $weight(p) = \alpha \sum_{u \in p} \frac{N(u)}{n} - (1-\alpha)\frac{|V_p|}{|V|}$; (Definition 1)
- $weight(p) = \alpha \sum_{u \in p} N(u)^x - y|V_p|$; for some $x, y \in \mathbb{R}^+$
- $weight(p) = \sum_{u \in p} e^{N(u)} - y|V_p|$; ($y \in \mathbb{R}^+$)
- $weight(p) = \sum_{u \in p} \log(N(u) + x) - y|V_p|$; ($x, y \in \mathbb{R}^+$)

4.1 MaxSum Top partitions (MST)

In this section, we intend to identify a set of k partitions, satisfying Property 4, that jointly have the highest weight among all combinations of k partitions. In other words, we aim to identify k completely important partitions (Property 4) such that the sum of their weights is maximized.

Problem 3. *(Max-Sum Top partitions) Let the function $weight(.)$: Partition $\rightarrow \mathbb{R}$ assigns a weight value to each partition. Moreover, let k be an input integer. Find k partitions $P_1, \cdots, P_k$ on the given lattice L such that:*

a. the objective $\Psi = \sum_{i=1}^{k} weight(P_i)$ is maximized
b. for each edge e in a partition: $weight(e) > 0$ (Lemma 1).

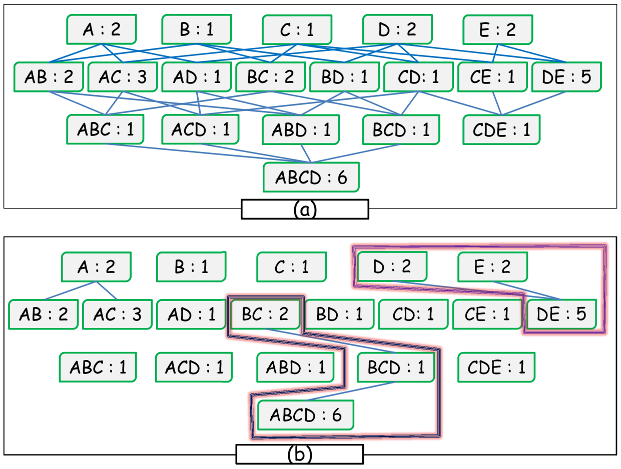


Figure 5: Finding 2 MST top partitions.

We present an algorithm that optimally solves Problem 3. We calculate the weights of edges in the lattice, remove all edges with negative weights and compute the weight of each connected component of the new lattice. We report k largest in weight components, provided there exist k components with a weight not smaller than the extra weight we achieve by cutting an edge ($\frac{1-\alpha}{|V|}$ according to Definition 1).

If the number of the components satisfying the condition is less than k (say, d), we create k partitions by segmenting the current components. This can be easily done by obtaining the spanning trees of these components and removing $k-d$ edges from them.[2] Note that the way, we select these edges, does not impact the value of the objective function. In case, the number of edges in all trees is smaller than $k-d$ (say d'), we report the trees together with $k-d-d'$ largest in weight components of the lattice having a weight smaller than $\frac{1-\alpha}{|V|}$.

Example 3. Consider the lattice in Figure 5(a). We wish to find top-2 partitions according to Problem 3. Let $\alpha = 0.5$. First, we calculate the weight of each edge in the lattice. The weight of the edge $e = (u,v) = 1/2 \times \frac{\min(N(u),N(v))}{n} - \frac{1}{2 \times |V|}$. In this lattice, $n = 35$ and $|V| = 19$. Thus, edges connected to nodes with size of 1 have negative weights and should be removed. Figure 5(b) displays the new lattice. The two partitions shown have the highest weights (0.0496) among all 13 candidates.

Theorem 7. *The proposed algorithm achieves the optimal solution for Problem 3.*

Theorem 8. *The run time of MST is $\Theta(l)$ where l is the number of lattice nodes.*

4.2 MaxWeighted Top partitions (MWT)

In the second ranking problem that we introduce, we wish to identify k partitions, satisfying Property 4, with the highest weights. Indeed, we are seeking k partitions that are as large as possible and report on the information captured by the tweets in them. Formally, the goal is to identify k completely important partitions (Property 4) such that the weight of the lightest one is maximized.

[2] Note that the weight we gain by removing an edge is $\frac{1-\alpha}{|V|}$

Problem 4. *(MaxWeighted Top k partitions) Let the function $weight(.) : Partition \rightarrow \mathbb{R}$ assigns a weight value to each partition. Moreover, let k be an input integer. Identify k partitions $P_1, \cdots, P_k$ in the given lattice L such that:*
a. The objective $\Phi = \min_{1 \leq i \leq k} weight(P_i)$ is maximized
b. for each edge e in a partition: $weight(e) > 0$ (Lemma 1)

Theorem 9. *Problem 4 is NP-complete.*

4.2.1 An Approximate MWT Algorithm (AMWT)

We start by pruning all edges with negative weights. Then, we calculate the weights of the connected components, and identify the k components with the largest weights.

These k components are not necessarily the top k partitions that we are looking for. There are situations where one can generate a set of k partitions with a larger minimum weight, by decomposing some of these components. In a set of partitions (for instance $\{P, Q, R\}$ where R is the lightest partition), a large partition (P) may be decomposed into two smaller partitions (P_1, P_2) with weights greater than the current minimum weight ($weight(P_1) > weight(R)$ and $weight(P_2) > weight(R)$). Hence, substituting P and R by the new partitions P_1 and P_2 increases the minimum weight in the set of partitions. This new set $\{P_1, Q, P_2\}$ has a higher minimum weight, hence it is a better solution.

To identify the top partitions, therefore, we should *optimally* decompose each component. The *optimal decomposition* for a component is a decomposition that increases the current minimum weight the most. To achieve this goal, we need to calculate the optimal number of decomposition for each component. We initialize this number to be 1 for all components. If there are less than k components (say h), we initialize this number to be $k - h + 1$ for the heaviest component and 1 for the others. These numbers are being updated in several iterations. In each iteration, we check whether a higher minimum weight can be achieved by increasing the number for one component and decreasing it for another. If so, we perform this change and continue until no increase can be done in the minimum weight. Note that calculating the optimal minimum weight when a component is decomposed into q partitions is, in fact, a MMP problem (Problem 2) and can be addressed utilizing any MMP algorithm including Algorithm 2 (by using a *weight* function instead of *size* in the pseudo code).

Theorem 10. *Utilizing a β-approximation algorithm for MMP, the approximation bound of AMWT is β.*

Theorem 11. *The run time of the AMWT algorithm (utilizing Algorithm 2 to conduct MMP) is $\Theta(k^2 \times l)$.*

4.2.2 A Heuristic MWT Algorithm (HMWT)

Arguably the runtime of AMWT can be high for large datasets or large k. We propose a much faster heuristic approach that generates partitions with minimum weight that is very close to the AMWT algorithm. Algorithm HMWT starts similarly to AMWT: it removes negative edges, identifies k connected components with the largest weights, and continues to decompose these components in several iterations. At each iteration, however, it first identifies the lightest current partition (P_o). Then, it temporarily decomposes each partition P_i ($1 \leq i \leq k$) into 2 parts P_i^1 and P_i^2. Let $MIN_i = \min(weight(P_i^1), weight(P_i^2))$ and $MAX = \max_{i=1}^{k}(MIN_i)$. Assume MAX corresponds to the partitions of P_j (i.e., $MAX = MIN_j$). HMWT compares MAX with $weight(P_o)$. If $MAX > weight(P_o)$, it replaces P_o and P_j with P_j^1 and P_j^2. Otherwise, the algorithm terminates and the k current partitions will be reported.

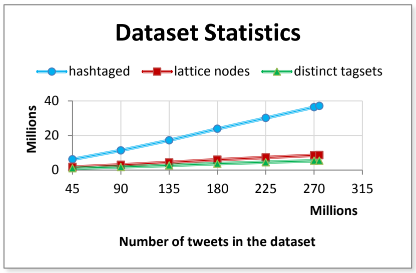


Figure 6: Dataset statistics.

We note that we do not need to temporarily decompose all partitions at each iteration. In fact, at iteration l, we reuse the values of MIN_i from the iteration $l - 1$ and run the MMP algorithm to decompose just the two newly generated partitions at iteration $l - 1$ (i.e., P_j^1 and P_j^2).

Theorem 12. *The run time of HMWT is $\Theta(k \times l)$.*

5. EXPERIMENTAL RESULTS

We evaluated all algorithms on synthetic and real datasets. Since the results are consistent in both datasets, we choose to present the results on real datasets.

We use a dataset consisting of 10 days of Twitter fire hose (chosen randomly from January and February 2012). Each day contains between $250 - 300$ million tweets, among those about 13% contain hashtags. We executed our algorithms on the different days and observed similar trends. The results on the run time of all proposed algorithms are presented in Section 5.1 and qualitative results of their output are discussed in Section 5.2.

The algorithms were coded in Java and evaluated on a quad core 2.4GHz computer (AMD Opteron^{TM} Processor 850) with 20G of memory running CentOS 5.5 (kernel version 2.6.18-194.11.1.el5). All algorithms are single-threaded.

5.1 Run time analysis

There are three parameters that affect the execution time of the partitioning algorithms: *(1)* the dataset size, *(2)* minimum size of partitions, c, in MSP, and *(3)* number of partitions, q, in MMP. The top-partitions algorithms are affected by three parameters: *(1)* dataset size, *(2)* number of partitions k, and *(3)* parameter α. To evaluate the impact of dataset size, since the tweets are ordered by time, we progressively increase the number of tweets. We use tweets for Feb 8 and report the number of tweets containing hashtags, the number of distinct tagsets, and the number of nodes in the corresponding lattice in Figure 6. As Figure 6 shows, Feb 8 contains more than 37 million tweets with hashtags, and about 5.5 million distinct tagsets. Moreover, the lattice representing this dataset contains about 8.7 million nodes. The lattice creation time takes less than 5 minutes for all algorithms utilizing our infrastructure. We analyze the run time for the partitioning algorithms in Section 5.1.1, and for the top-partitions algorithms in Section 5.1.2.

5.1.1 Partitioning algorithms

Figure 7 depicts the sensitivity of the partitioning algorithms to the dataset size and the parameter q. We set c=q=100 in Figures 7(a-b). Figures 7(a-b) suggest that the

Table 1: Run time of all algorithms for c=q=k=100, α=0.25 on Feb 8 tweets.

Alg.	AMSP	MMP	MST	AMWT	HMWT
Time	9.3 sec	18.7 min	9.2 sec	2.9 min	32.9 sec

run time of AMSP and MMP increases linearly when the dataset size grows. We, also, measure the impact of changing c (min size of partitions) on the run time of the AMSP algorithm. Our experiments show that the run time remains constant (about 9.3 seconds) when c increases; changing c does not affect the number of times the lattice nodes and edges are examined by AMSP. Thus, AMSP can generate partitions on a full hose tweet dataset with any desired minimum size in less than 10 seconds. As Figure 7(b) shows, the MMP algorithm, partitions the entire dataset into 100 partitions in less than 19 minutes. In Figure 7(c), we measure the run time of MMP by varying the value of q and observe a linear increase in the run time as q increases.

5.1.2 *Top-partitions algorithms*

In Figure 8, we demonstrate the run time behavior of the top-partitions algorithms by varying the dataset size and the number of partitions, k. We set $k = 100$ in Figures 8(a-b) and run the MWT algorithms on the full dataset in Figures 8(c-d). We observe a linear run time increase as a result of dataset growth (Figure 8(a-b)). This linearity persists when k is raised from 10 to 1 million in HMWT (Figure 8 (d)). Figure 8(c) demonstrates that AMWT has a quadratic run time subject to k (as discussed in Section 4.2.2).

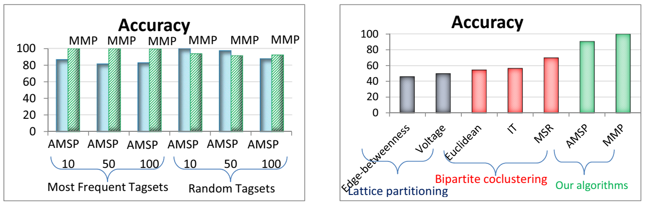
Figure 9 demonstrates the impact of parameter α on the run time of the top-partitions algorithms. The results for MST have been presented in Figure 9(a) and for the MWT algorithms in Figure 9(b). We observe a slight increase in the run time of MST for $\alpha = 0.8$ that is reasonable since the number of non-negative edges increases significantly at that point. To explain the behavior of MWT in Figure 9(b), we show how the number of non-negative edges vary when α increases (Figure 9(c)) and how run time changes with respect to the number of non-negative edges (Figure 9(d)). The outcome suggests a linear increase in run time with respect to the number of non-negative edges.

We report the run time for all algorithms on the full dataset of Feb 8 using c=q=k=100 and α=0.25 in Table 1.

5.2 Qualitative results

In this section, we conduct a qualitative study of the results of all algorithms presented. As a baseline comparison, we include two graph clustering algorithms to partition the lattice and three co-clustering algorithms to partition the tweet-tag bipartite graph. The tweet-tag bipartite graph for a collection of tweets is constructed by establishing edges between tweets (as nodes on one partition) and the hashtags contained in the tweets (as nodes on the other partition). Moreover, we compare our ranking algorithms with an algorithm that simply ranks tagsets based on their frequency (referred to as algorithm MFT - Most Frequent Tagsets) which is further compared in Section 5.2.1).

After these comparisons, we report results of running our algorithms on the Twitter fire hose for Feb 26 (The Oscars night) using c=q=k=100 and α=0.001 for MST and α=0.25 for MWT algorithms. Finally, we analyze how the partitions alter when we change the values of different parameters.



(a) partitioning quality

(b) baseline comparison

Figure 10: Accuracy comparison.

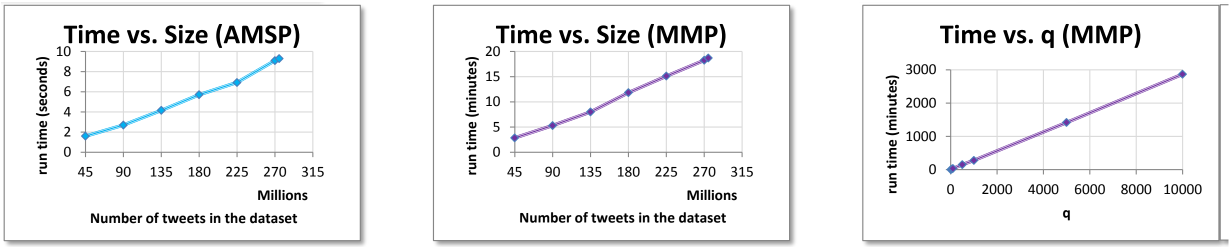
To report the results, we rank them accordingly. For MSP, MST, and MWT, due to the constraints and objectives on both the size and the number of edges in each partition, we use the weight function defined in Definition 1 to rank the generated partitions. For MMP, since the goal is to maximize the size of each partition with no constraint on the number of edges, we rank them according to their sizes.

5.2.1 *Goodness of the proposed algorithms*

To evaluate our partitioning algorithms, we created a set of datasets containing known partitions. We experimented with various ways to generate known partitions including among others, selecting the partitions randomly, selecting the partitions based on what was trending that day on Twitter as well as selecting partitions based on very popular hashtags on that day. In all cases our results were consistent. For brevity we report the following indicative results. We choose the 10, 50, 100 most frequent tagsets and 10, 50, 100 random tagsets and in each case retrieve all tweets in the Twitter fire hose containing at least one of these tagsets. Here, each tagset represents a partition. We run AMSP and MMP to segment these datasets into 10, 50, 100 partitions and report the accuracy (the percentage of correctly partitioned tweets) in Figure 10(a). As can be seen our algorithms are highly accurate.

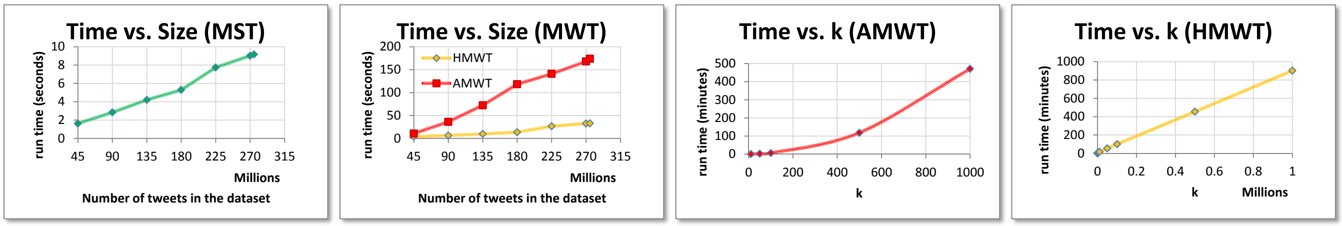
We also compare the results of our partitioning algorithms with baseline techniques including two graph clustering algorithms (the edge betweenness clustering [6] and the voltage clustering [17]) on the created lattice and three co-clustering algorithms (the euclidean co-clustering, information theoretic co-clustering "IT", and minimum squared residue co-clustering "MSR") [3, 4, 7] on the tweet-tag bipartite graph. Since the complexity of these algorithms is high, we couldn't run them on the full dataset. For exposition, we run all algorithms on a smaller dataset created by choosing the 10 most frequent tagsets and retrieving 1000 first (by post time) tweets containing each tagset (10000 tweets in total). The edge betweenness clustering (as an example) takes 4 days on this dataset while AMSP and MMP take less than 0.2 seconds. Figure 10(b) displays the results. It is evident that our approach offer superior accuracy in all cases. Moreover in terms of run time our approach is the only approach that is able to scale into real world instances such as Twitter.

To evaluate our ranking algorithms, we discuss a final way to report top tagsets, namely the strategy MFT that simply reports the most frequent tagsets. There are significant differences between the results of this approach and those discovered by our algorithms. People commonly employ diverse tagsets to refer to the same physical event. MFT does not group related tagsets in one partition hence fails to detect all important events or rank them effectively. Further-



(a) Time vs. size (AMSP) (b) Time vs. size (MMP) (c) Time vs. q (MMP)

Figure 7: The impact of changing size and q on the run time of the partitioning algorithms.



(a) Time vs. Size (MST) (b) Time vs. Size (MWT) (c) Time vs. k (AMWT) (d) Time vs. k (HMWT)

Figure 8: The impact of changing size and q on the run time of the top-partitions algorithms.

more, each MFT partition is a single tagset hence it does not discover context. More details are provided in [5].

5.2.2 Detected trends

Table 2 presents the top partitions for each algorithm. Since the partitions contain diverse tagsets, to ease our presentation and facilitate reference, we label each partition utilizing the most frequent tagset in it and show the partition labels using capital letters (e.g., #OSCARS).

By inspecting Table 2, we observe that many partition labels are shared among the algorithms. Some observations are in order; we note that the Oscars 2012, the 2012 NBA All-Star game between Eastern and Western conference teams, the League Cup final (Carling Cup) between Liverpool and Cardiff (partition labels #LFC, and #YNWA[3]), a derby match between Arsenal and Tottenham, soccer matches of Rayo Vallecano vs Real Madrid (#HALAMADRID), PSV Eindhoven vs Feyenoord (#PSVFEY), a government referendum on a new constitution in Syria, and the national dutch songfestival all took place on Feb 26. Moreover, we realized that there was active discussion regarding #LIBROQUEESTOYLEYENDO (in Spanish: Book I'm reading), #10THINGSILOVEABOUTYOU, #THINGSPEOPLEHAVETOSTOPDOING, #TEECHANDEMISA (in Spanish: you get kicked out of church [if ...]), #5GOODSINGERS, #10VICIOS, #IFMYLIFEWEREAMOVIE, #BLACKMOMSCATCHPHRASE, and #5THINGSILOVEABOUTYOU.

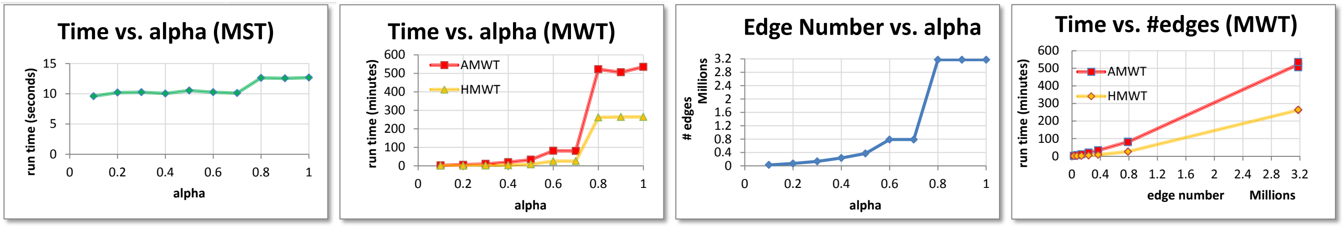
Furthermore, we observed other popular events regarding the partition with label #DAYTONA500, "a 500 miles (804.7 km)-long NASCAR Sprint Cup Series race held annually at the Daytona International Speedway" that was held in Feb 27-28, 2012[4]; the partition with label #ELDIARIODENOA, a Spanish name for the movie "The Notebook", that was broadcasted on Antenna 3 in Feb 26 in Spain and attracted social media attention; the partition with label #SARAHGLIVE that is "a Philippine evening musical show hosted by Sarah Geronimo [...] debuting on February 26, 2012"[5]; the partition with label #TVDMEMORIES about the TV series "the vampire diaries"; the partition with label #MYWORLDTOURMEMORIES about Justin Bieber's first headlining tour diaries that appears in fans' tweets about their favorite moments from the tour; the partition with label #BIGBANGFANTASTICBABY that appeared due to the release of a teaser for one of the Big Bang (a popular Korean band) new songs "Fantastic Baby", and the partition with label #WEMISSSHINEE related to the South Korean R&B boy group Shinee.

The similarity of the partition labels, representing the top partitions in different algorithms in Table 2, demonstrates the consistency of these algorithms in the experiments. Moreover, by manual inspection of the content of the partitions with similar labels, we observed that they are indeed related to the same physical events. Yet, we observe some differences in the ranking of these partitions among the presented algorithms. In fact, different algorithms yield partitions with labels at varying positions in the ranking on Table 2. For instance, in AMSP, the partition with label #TEAMEAST appears higher in the ranking than #SONGFESTIVAL while, in MMP, it appears lower. In fact, the number of tweets in the partition with label #SONGFESTIVAL is more than the partition #TEAMEAST. Recall that we rank MMP partitions according to their overall size. Hence, the partition labeled #SONGFESTIVAL appears higher than #TEAMEAST in the MMP ranking. On the other hand, the number of nodes (distinct tagsets) in the partition with label #TEAMEAST is less than the partition with label #SONGFESTIVAL. This results in a lower weight (Definition 1) for #SONGFESTIVAL, hence a lower position in the AMSP partitions' ranking. Another reason for this ranking difference is that although partitions with the same label, created by various algorithms, relate to the same physical event, it does not necessarily mean that they contain exactly the same set of tweets. Thus partitions may have diverse

[3] #lfc, a short for "Liverpool football club"; #ynwa, a short for "You'll never walk alone", the official Liverpool football club anthem, which is performed by fans on matchdays.

[4] http://en.wikipedia.org/wiki/Daytona_500

[5] http://en.wikipedia.org/wiki/Sarah_G._Live



(a) Time vs. α (MST) (b) Time vs. α (MWT) (c) Edge Number vs. α (d) Time vs. #edges (MWT)

Figure 9: The impact of changing α on the run time of the top-partitions algorithms.

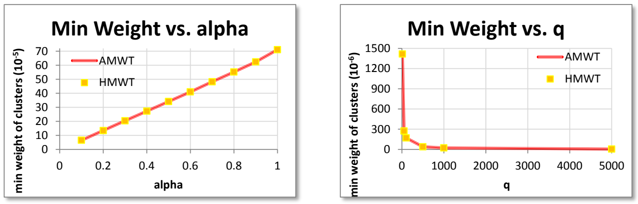
sizes and weights that can lead to differences in the ranking position. Section 5.2.3 discusses the structural differences among the partitions generated by the proposed algorithms.

5.2.3 *Differences in the algorithms*

Comparing MSP with the other algorithms, we realize that MSP creates partitions about specific events (exhibiting very low variance in the tagsets belonging to the partitions) while other algorithms identify events with much higher variance in the tagsets included in the partition. For example, the partition labeled `#OSCARS` in MSP contains tweets mostly tagged individually with a single `#oscars` along with a small number of tweets tagged with a few other tagsets such as (`#oscars #wish`) and (`#oscars #scam`) in 5 tweets and (`#oscars #videogames`) in 7 tweets. The partition with label `#OSCARS` in other algorithms contains several Oscars related sub-events. For instance, we can see hashtags in this partition for sub-events including Oscars movies (e.g., `#theartist`, `#hugo`, `#aseparation`, `#theironlady`), red carpet (`#redcarpet`), best and worst dressed (`#bestdressed`, `#worstdressed`), and the simultaneous occurrence of Oscars with NBA all-stars (`#oscars #allstar`).

To understand the differences between MMP and top-partitions algorithms, we first define the notion of a connector between partitions in the lattice. A connector is a tagset that builds a path between two partitions in the lattice. In other words, it is a tagset that is a superset/subset of at least one tagset in each partition with a difference in size exactly one. For example, amongst partitions `#a` and `#b`, (`#a #b`) is a connector since it is a superset of `#a` in the first partition and `#b` in the second. We say that a connector is **weak** if its size (number of tweets associated with this tagset) is much smaller than the corresponding size of the partitions it connects and **strong** otherwise. MMP allows the creation of large partitions with weak connectors while top-partitions algorithms create partitions containing tagsets associated via strong connectors (by removing negative edges). For instance, for the NBA all-star match that took place between the Eastern and Western conference teams, MMP creates one partition containing hashtags {`#nba`, `#allstar`, `#teameast`, and `#teamwest`}, while in top-partitions algorithms, we see two separate partitions with hashtags {`#nba`, `#allstar`} and hashtags {`#teameast`, `#teamwest`}. As the potential connectors of these two partitions are weak, the top partitions algorithms do not merge the two partitions.[6]

The partitions, created by MST, greatly depend on the value of α. MST removes edges depending on the value of α and selects the largest components in the remaining lattice. Since there is no limitation on the size of these partitions, it may create a set of partitions with heterogeneous sizes. Thus, we may observe large partitions encompassing different tagsets along with tiny partitions. For example, when $\alpha = 0.5$, we obtain a large partition containing tweets about diverse tagsets related to Oscars, NBA, soccer, etc., with more than 25 million tweets, and some small partitions (e.g., a partition with label `#10THINGSIHATE` with 12000 tweets). Choosing an appropriate α value is considerably important to achieve meaningful partitions. We explain how to choose an appropriate value for α in Section 5.2.4.



(a) min-weight vs. α (MWT) (b) min-weight vs. k (MWT)

Figure 11: The impact of changing q and α on the minimum weight of the proposed algorithms.

Finally, we focus on comparing the top partitions generated by the two algorithms presented for the MWT problem. Although in theory there could be scenarios where there is a difference between the partitions, on real data we observed similar behavior for both algorithms. We report the minimum weight of partitions that are generated by the AMWT and HMWT algorithms in Figure 11. We observe that HMWT can achieve the same minimum weight as AMWT in all except two cases and the difference between the reported values for these two cases is less than 0.04%. This suggests that the HMWT algorithm creates partitions with high quality (same as AMWT) albeit much faster.

5.2.4 *Choice of parameters*

We detail how the partitions alter when we change the parameters associated with our algorithms, and also what constitutes good parameter choice for the algorithms.

In MSP, increasing c (minimum partition size) results in the creation of larger partitions. For instance, by setting c to 100, we observe a partition with label (`#OSCARS #ASEPARATION`) for the Oscars movie "A Separation", while by increasing c to 1000, this partition becomes part of a larger partition containing information about several Oscars movies. On the contrary, increasing q (number of partitions), in MMP, leads to partitions representing smaller events. Example of this behavior is a big Oscars partition for $q = 10$ that is divided into several smaller partitions with la-

[6] The size of (`#nba #teameast`) and (`#nba #teamwest`) is, respectively, 58 and 28. Compare it with `#teameast` : 58097, `#teamwest` : 36152, (`#teameast #teamwest`) : 3968, `#allstar` : 65179, `#nba` : 13623 and (`#allstar #nba`) : 3230.

Table 2: The top results for different algorithms.[7]

rank	AMSP	MMP	MST	AMWT	HMWT
1	#OSCARS	#OSCARS	#OSCARS	#OSCARS	#OSCARS
2	#TPHTSD*	#TPHTSD*	#TPHTSD*	#TPHTSD*	#TPHTSD*
3	#10TILAY*	#10TILAY*	#10TILAY*	#10TILAY*	#10TILAY*
4	#10VICIOS	#ALLSTARGAME	#YNWA	#ALLSTARGAME	#ALLSTARGAME
5	#BMCP*	#BMCP*	#BMCP*	#BMCP*	#BMCP*
6	#ALLSTARGAME	#10VICIOS	#10VICIOS	#10VICIOS	#10VICIOS
7	#YNWA	#LFC	#TEAMEAST	#LFC	#LFC
8	#5GOODSINGERS	#ARSENAL	#5GOODSINGERS	#ARSENAL	#ARSENAL
9	#MWTM*	#YNWA	#MWTM*	#YNWA	#YNWA
10	#TEAMEAST	#SONGFESTIVAL	#SONGFESTIVAL	#TEAMEAST	#TEAMEAST
11	#ARSENAL	#TEAMEAST	#ARSENAL	#5GOODSINGERS	#5GOODSINGERS
12	#SONGFESTIVAL	#5GOODSINGERS	#HALAMADRID	#SONGFESTIVAL	#SONGFESTIVAL
13	#LFC	#SYRIA	#TEECHANDEMISA	#MWTM*	#MWTM*
14	#TEECHANDEMISA	#MWTM*	#DAYTONA500	#SYRIA	#SYRIA
15	#WEMISSSHINEE	#EREDCARPET	#WEMISSSHINEE	#EREDCARPET	#EREDCARPET
16	#EREDCARPET	#ACADEMYAWARDS	#ELDIARIODENOA	#ACADEMYAWARDS	#ACADEMYAWARDS
17	#ELDIARIODENOA	#TEECHANDEMISA	#TVDMEMORIES	#TEECHANDEMISA	#TEECHANDEMISA
18	#ACADEMYAWARDS	#WEMISSSHINEE	#OSCARLINHOSBROWN	#HALAMADRID	#HALAMADRID
19	#TVDMEMORIES	#TEAMWEST	#SARAHGLIVE	#WEMISSSHINEE	#WEMISSSHINEE
20	#TEAMWEST	#DAYTONA500	#SYRIA	#DAYTONA500	#DAYTONA500
21	#IMLWAM*	#IMLWAM*	#OSCARFORPOTTER	#ELDIARIODENOA	#ELDIARIODENOA
22	#SARAHGLIVE	#SARAHGLIVE	#5TILAY*	#TEAMWEST	#TEAMWEST
23	#OSCARLINHOSBROWN	#TVDMEMORIES	#BBFB*	#IMLWAM*	#IMLWAM*
24	#OSCARFORPOTTER	#PSFVEY	#MOVIESWEREBESTWHEN	#SARAHGLIVE	#SARAHGLIVE
25	#SYRIA	#HALAMADRID	#PSVFEY	#TVDMEMORIES	#TVDMEMORIES

bel (#REDCARPET), (#BESTDRESSED), (#THEARTIST), (#HUGO), etc. when $q = 1000$. What constitutes a good choice of c and q depends on our intention to obtain large partitions or small specific partitions. A suggested approach is to choose large values for c and decrease it progressively to obtain more refined partitions. By decreasing c, the MSP algorithm generates partitions for popular events in addition to partitions representing smaller events. The same approach can be utilized for q where we start by choosing a small value and increase it to acquire more detailed partitions as well as partitions representing less popular events.

The parameter α, as mentioned in Section 4, specifies the trade-off between having more tweets or less distinct tagsets when partitions are created. When we decrease α, we remove many edges from the lattice. Thus, we obtain smaller in size partitions representing more specific events. When $\alpha = 0$, the top partitions are the most frequent tagsets, while when $\alpha = 1$, the top partition is the largest connected component of the lattice. We run the MST algorithm using different values of α and observe the created partitions. Recall that in order to identify top partitions in MST, we choose the largest connected components of the lattice. When the value of α is large, a small number of edges is removed and most of the tagsets are still connected. For instance, when $\alpha = 0.5$, the top partition has 25 million tweets merging different tagsets #oscars, #allstar, #arsenal, #fb, #oomf, #10thingsiloveaboutyou, #bahrain, #syria, #china, etc. The remaining top partitions are connected components, representing small events. For example, we observe a partition with label #ZACEFRONNAJOHNJOHN (Zac Efron, the American actor and singer, that was announced as the face of John John denim) with 23500 tweets, and a partition with label #10THINGSIHATE with 12000 tweets.

When we decrease the value of α, more edges are removed. Thus, the connected components break to smaller pieces. Therefore, if the connectors between partitions are weak, they will be separated. For instance, when $\alpha = 0.01$, the top partition contains tweets with tagsets #nowplaying, #nowwatching, #oscars, #news, #carlingcup-liverpool, and #nba. The second partition contains #10thingsiloveaboutyou, and the third partition contains tweets about Arab countries with tagsets #bahrain, #kuwait, #saudi, #uae and #syria. When $\alpha = 0.001$, the top partitions contain more specific events (Table 2). In this case, the top partition represents the Oscars event including tweets about Oscars sub-events such as Oscars, red carpet, best dressed, winning movies (e.g., the artist, the help, Hugo), and its co-occurrence with the all-star game.[8] It is interesting to observe how this partition morphs when we further decrease α. By reducing α, this partition is divided into a number of sub-partitions. Hence, we get more refined partitions related to Oscars. For instance, for $\alpha = 0.0005$, the top partition includes Oscars, red carpet, best dressed, and the winning movies (the artist, Hugo). This is a more refined Oscars partition that does not contain tweets related to the co-occurrence of Oscars and the NBA all-star game. Moreover, it just contains the most popular (most tweeted about and incidentally receiving the most awards) movies of the Oscars, the artist and Hugo. When we further decrease α to 0.0002, we achieve a very refined top partition containing Oscars and red carpet tweets. Finally by decreasing α to 0.0001, the top partition contains tweets tagged solely with #oscars.

[7] We have used #TPHTSD, #10TILAY, #BMCP, #MWTM, #IMLWAM, #5TILAY, and #BBFB as abbreviations for #THINGSPEOPLEHAVETOSTOPDOING, #10THINGSILOVEABOUTYOU, #BLACKMOMSCATCHPHRASE, #MYWORLDTOURMEMORIES, #IFMYLIFEWEREAMOVIE, #5THINGSILOVEABOUTYOU, and #BIGBANGFANTASTICBABY.

[8] A strong connection exists between Oscars and all-star game due to their co-occurrence. Many tweets complain here and talk about which one to watch, and how this co-occurrence may reduce the revenue that can be achieved by their organizers.

This points to an iterative approach to obtain meaningful partitions; one can start by using a high value for α and progressively decrease it observing the refined partitions. In general, if we use the proposed weight function and desire partitions containing tagsets with sizes of at least a, we should choose $\alpha = \frac{n}{n+a\times l}$ where n and l are, respectively, the number of tweets and nodes in the lattice.

6. RELATED WORK

The use of tags in online information sharing has been utilized in various contexts. For example collaborative tagging has been utilized to solve different data mining problems. Heyman et al. [8] study whether social annotations (bookmarking and tagging) can provide data to improve web search. Sarkas et al. [16] have designed novel searching paradigms employing user-generated tags Hotho et al. [9] identify trends in a social tagging system (folksonomy) utilizing a differential adaptation of the PageRank algorithm. Li et al. [12] create a top-down hierarchical structure to store tags. An automatic method to recommend personalized tags for webpages has also been designed by Chirita et al. [2].

Twitter has been considered actively in research studies given the relatively easier accessibility to its data sets. Sankaranarayanan et al. [15] present a system to detect breaking news in twitter. Other aspects of research related to Twitter are available elsewhere [11]. Several works [10,13] aims to characterize bursty information on the web. Kleinberg [10] utilizes a multi-state automaton to model and identify the bursty vs. non-bursty time periods for each event. Mathioudakis et al. [13] detect geographically focused information bursts and describe these bursts with demographic characteristics and illustrative keywords. Budak et al. [1] utilize the network topology to discriminate viral bursty topics from news media bursty topics.

Perl and Schach [14] have proposed an optimal algorithm to divide a tree into q subtrees (by removing $q-1$ edges) such that the size of the lightest subtree is maximized. In Section 3.2 we generalize this approach to partition a lattice.

Graph clustering algorithms have been proposed based on various concepts such as analogies from physics (voltage clustering [17]) or concepts utilizing graph notions (such as sorest paths, edge betweenness [6]) or simultaneous similarity of nodes and features (co-clustering approach [3, 4, 7]). We compare with those in section 5.2.1 for completeness. We note however that the objectives of these algorithms are orthogonal to those of our problems and in terms of run time they take hours to complete even in small instances of our problem (e.g., running the edge-betweenness clustering algorithm on the same data set as that used in Section 5.2.1 with 10000 tagged tweets takes approximately 4 days).

7. CONCLUSIONS

We presented several algorithms to partition and rank tagged data sources and also investigated the complexity of these algorithms. All algorithms were evaluated on the Twitter fire hose and their performance was demonstrated by varying all key parameters. We presented a detailed quantitative and qualitative analysis. Our algorithms operate on the full day of Twitter data in seconds typically and in less than 20 minutes for the most time-consuming algorithm proposed. We also demonstrated that the top partitions created by our algorithms correspond to valid popular events of the day. Furthermore, we compared our algorithms with 5 baseline techniques (lattice partitioning and bipartite tweet-tag coclustering) and noticed up to 55% improvement in accuracy over baseline techniques.

This work raises several avenues for further work. One is to design algorithms to improve the approximation bounds. It would be interesting to incorporate more information including time, space, and social ties in the creation of the lattice model.

8. REFERENCES

[1] C. Budak, D. Agrawal, and A. El Abbadi. Structural trend analysis for online social networks. *VLDB Endowment*, 4(10):646–656, 2011.

[2] P. A. Chirita, S. Costache, W. Nejdl, and S. Handschuh. P-tag: large scale automatic generation of personalized annotation tags for the web. WWW, pages 845–854, 2007.

[3] I. Dhillon and Y. Guan. Information theoretic clustering of sparse co-occurrence data. ICDM, pages 517–520, 2003.

[4] I. S. Dhillon, S. Mallela, and D. S. Modha. Information theoretic co-clustering. SIGKDD, pages 89–98, 2003.

[5] M. Eftekhar and N. Koudas. Partitioning and ranking tagged data sources. Technical Report TR–DM–UT–12–05–00, University of Toronto, 2012.

[6] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

[7] Y. G. H. Cho, I.S. Dhillon and S. Sra. Minimum sum-squared residue co-clustering of gene expression data. SDM, pages 114–125, 2004.

[8] P. Heymann, G. Koutrika, and H. Garcia-Molina. Can social bookmarking improve web search? WSDM, pages 195–206, 2008.

[9] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Trend detection in folksonomies. SAMT, pages 56–70, 2006.

[10] J. Kleinberg. Bursty and hierarchical structure in streams. SIGKDD, pages 91–101, 2002.

[11] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? WWW, pages 591–600, 2010.

[12] R. Li, S. Bao, Y. Yu, B. Fei, and Z. Su. Towards effective browsing of large scale social annotations. WWW, pages 943–952, 2007.

[13] M. Mathioudakis, N. Bansal, and N. Koudas. Identifying, attributing and describing spatial bursts. *VLDB Endowment*, 3(1-2):1091–1102, 2010.

[14] Y. Perl and S. R. Schach. Max-min tree partitioning. *Journal of the ACM*, 28(1):5–15, 1981.

[15] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. GIS, pages 42–51, 2009.

[16] N. Sarkas, G. Das, and N. Koudas. Improved search for socially annotated data. *VLDB Endowment*, 2(1):778–789, 2009.

[17] F. Wu and B. Huberman. Finding communities in linear time: a physics approach. *European Physical Journal B*, 38(2):331–338, 2004.