Information streams on the web

Dan VODISLAV

CY Cergy Paris Université

Master Recherche M2 SIC

Databases and streams

- Data streams
  - Data arriving in streams, with temporal stamps
  - E.g. stock quotes, news, sensor data (position, temperature, …), social network messages, etc.
- What changes relative to classical databases?
  - One may see each data item as a new line in a database table
  - Arrival of a new item = insertion into the table

… but

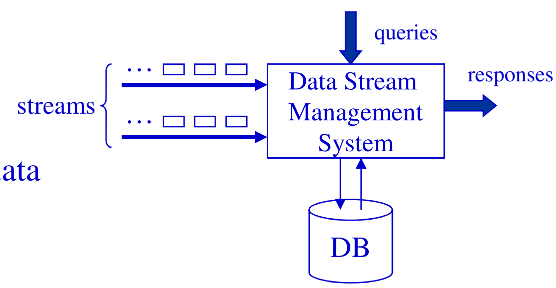
- One cannot store everything
- Need to react on the arrival of a new item
- Time plays a particular role
  - At the data level: time stamp, item order
  - At the event level: moment when a new item arrives

Why stream processing?

- Volume of data
  - Impossible to store everything → Big Data
- Frequent production of new content
  - Reduce the delay between content production and consumption → react on the arrival of a new item
- Many datas are naturally produced as streams
  - Sensors, stock quotes, news headlines, …
  - Applications for monitoring, surveillance, watching
- Accelerated processing
  - On-the-fly processing, filtering based
  - Reduced need for (slow) access to stored data

Stream processing approaches

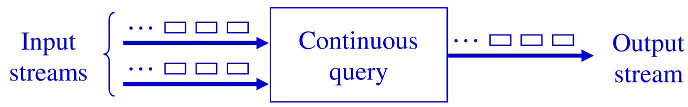
- Two extreme approaches
  - Store everything (static)
    - Every new arrived item is stored
    - Static (snapshot) query on the stored data
  - Continuous processing (dynamic)
    - No storage
    - Continuous queries



- Intermediary approaches may be imagined
  - Store everything, but trigger snapshot queries on the arrival of new data
    - E.g. A snapshot query every N new items
  - Continuous processing, but accessing stored data
    - E.g. Filter the new items on a criteria based on stored data
- Stream types
  - Data streams: items = structured data
  - Information streams: items = text

Continuous queries

- Classical queries: data + queries at different moments
  - Processing done at query time, on current data
- Continuous queries: query + data at different moments
  - Processing done each time new data arrives
- Specificities of continuous queries
  - Number of results – undetermined
  - No access to the whole data
    - Current item + possibly some older items stored by the system
  - Results produced only on input events (arrival of an item)
  - Generally less complex than classical queries
- Continuous query = *subscription*
  - A result → *notification*
- General model



Windows on streams

- Window = finite subsequence of stream items
  - Preserve the order of the input items
- Inherent to stream processing
  - Storage: actually, *we store windows on streams* (streams are unbounded)
  - Continuous queries: necessary for expressing joins between streams
  - Specific operations on streams: *aggregation* on windows

    E.g.. The average of the last 10 days stock quotes
- Window types
  - *Sliding*: determined by the current moment
    - On duration: the items of the last *n* time units (hours, days, etc.)
    - On the number of items: the last *n* items
  - *Condition based*: determined by begin/end conditions

    E.g. Starts when the quote becomes smaller than 40 $, ends by the end of the day
  - *Tumbling*: partition items in a fixed way (by day, week, etc.)

Information streams

- Streams of text messages
  - Web syndication channels
  - Social network messages
- Web syndication
  - New information published on a communication channel
  - Channel → content periodically updated by the server → information stream
  - Users subscribe to channels
- RSS (Rich Site Summary, Really Simple Syndication)
  - XML format for information publishing on the web
  - Publication by updating an XML file that contains the most recent items
    - Updated XML file → information stream

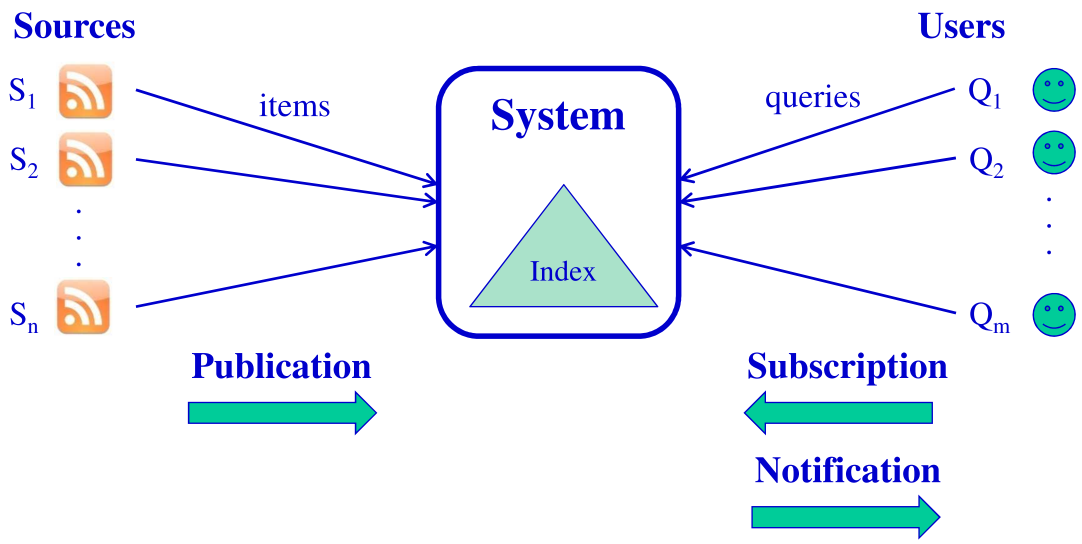
RSS example

```
<?xml version='1.0' encoding='UTF-8' ?>
<rss version="2.0">
  <channel>
    <title>Le Monde.fr : Actualités a la une</title>
    <link>http ://www.lemonde.fr</link>
    <language>en</language>
    <copyright>Copyright Le Monde.fr</copyright>
    <pubDate>Fri, 11 Apr 2008 13 :36 :10 GMT</pubDate>

    <item>
      <title>Faute de "réponses concrètes", la FIDL appelle a une nouvelle grève mardi</title>
      <link>http ://rss.feedsportal.com/c/205/f/3050/s/e2301b/story01.htm</link>
      <description>Le syndicat lycéen, reçu vendredi par le ministre de l'éducation, n'a pas été convaincu par les arguments de
Xavier Darcos. L'UNL, premier syndicat lycéen, doit être reçu en fin d'après-midi par le ministre.</description>
      <pubDate>Fri, 11 Apr 2008 13 :24 :10 GMT</pubDate>
      <guid isPermaLink="false">http ://www.lemonde.fr/societe/article/2008/04/11/
faute-de-reponse-concrete-la-dl 1033455 3224.html ?xtor=RSS-3208</guid>
    </item>

    <item>
      <title>Le Conseil d'Etat consacre le secret professionnel des avocats</title>
      <link>http ://rss.feedsportal.com/c/205/f/3050/s/e2301c/story01.htm</link>
      <description>La haute juridiction administrative a annulé partiellement, jeudi 10 avril, le décret d'application de la deuxième
directive européenne contre le blanchiment des capitaux.</description>
      <pubDate>Fri, 11 Apr 2008 13 :10 :44 GMT</pubDate>
    </item>
    ...
```

Publish-subscribe systems



- An item published by a source may interest several queries
  - The index allows efficiently targeting the queries for notification

Example of index

- Boolean text queries
  - Common case: conjunctive queries = set of keywords
  - Goal: find the items containing all the keywords of the query
- Index = inverted file
  - Built from the queries (the set of all the words in the queries)
  - Keyword m → set of queries q containing the keyword

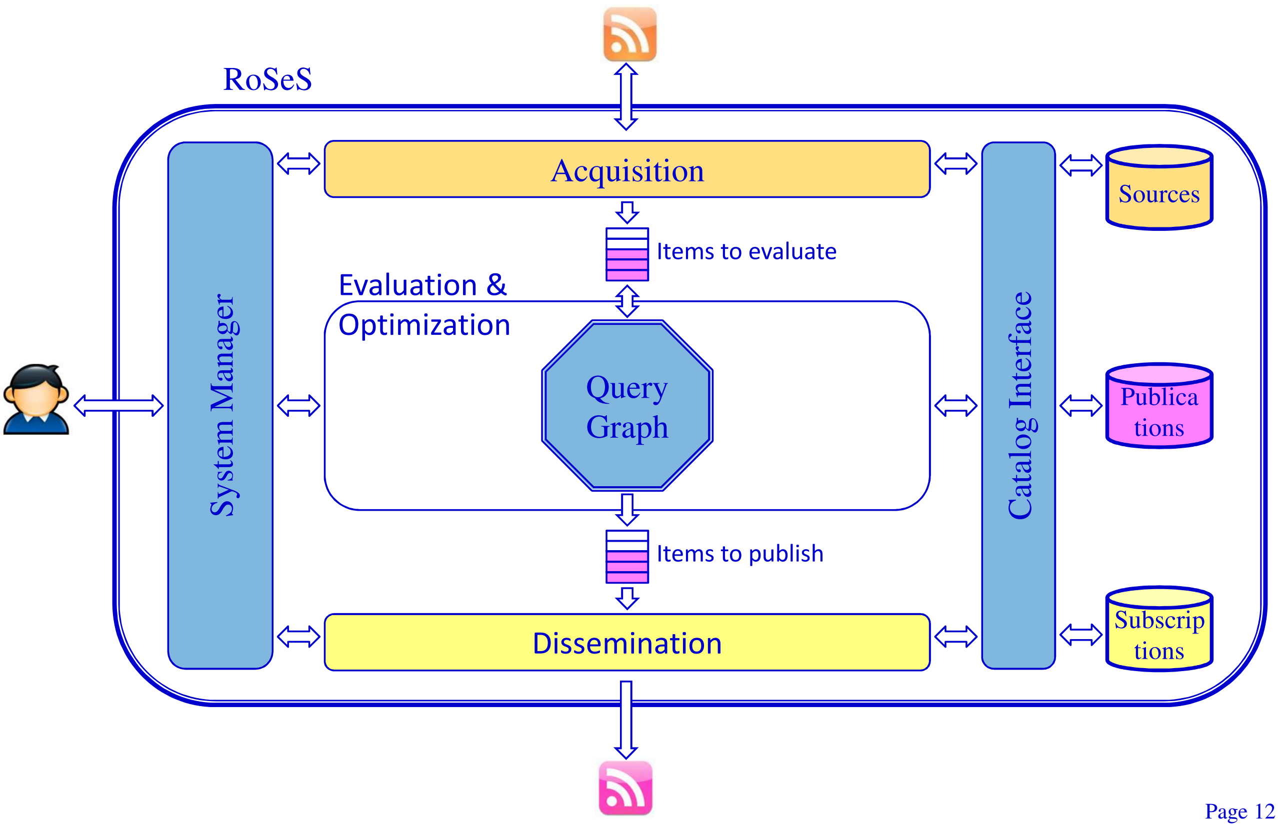
$m_1 \rightarrow q_{1,1}; q_{1,2}; \ldots$
$m_2 \rightarrow q_{2,1}; q_{2,2}; \ldots$
…
$m_n \rightarrow q_{n,1}; q_{n,2}; \ldots$

- How it works
  - Item *i* published
  - For each keyword *m* of *i* : we get the list of queries *index*(*m*)
  - For each query in *index*(*m*), we increment the number of found keywords
  - Result: the queries that find all their keywords in *i*

Example of an information stream aggregator

- RoSeS
  - Research system for RSS information stream aggregation
  - More complex queries: keyword filtering, union, join
  - Multi-user publish/subscribe system
  - Continuous processing of the queries, no storage
  - Stream personalization and sharing

Architecture



The RoSeS language

3 types of instructions

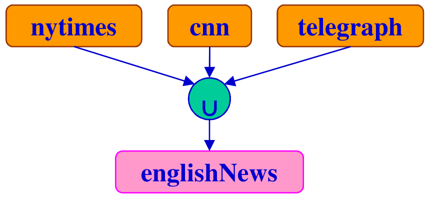
- **register feed**
  - Defines internal names for Source streams

  e.g.: **register feed** "http://feeds.nytimes.com/nyt/rss/HomePage" **as** nytimes

- **create feed**
  - Creates new streams (Publications)

  e.g.: **create feed** englishNews

  from nytimes | cnn | telegraph



- **subscribe to**
  - Defines a Subscription to a Publication, a notification mode (rss, mail, sms) and a notification frequency

  e.g.: **subscribe to** englishNews **output mail** "Jordi.Creus@lip6.fr" **every** 12 **hours**

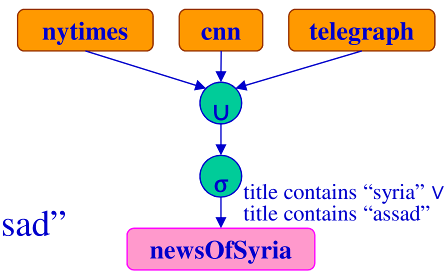
Query (publication) language

- 4 operators: union, selection, join & window
- We want to express in the language
  - Big unions on collections of streams
  - Apply text filters on unions
  - Associate items from several streams (join)
- *Example 1*

  create feed newsOfSyria

  from nytimes | cnn | telegraph

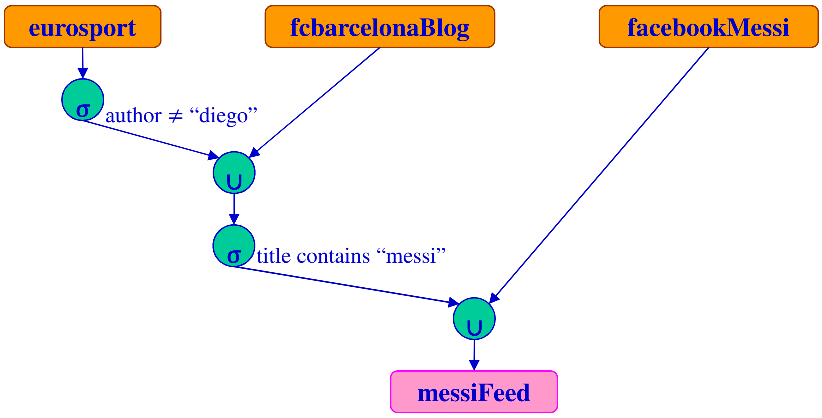
  where title **contains** "syria" **or** title **contains** "assad"



Query (publication) language

- *Example 2*

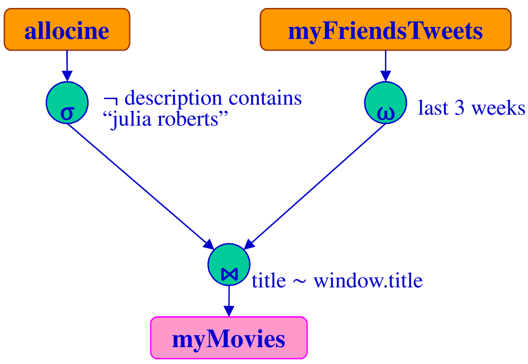
create feed messiFeed
from (eurosport **as** $e | fcbarcelonaBlog) **as** $u | facebookMessi
where $e[author <> "diego"] **and**
$u[title **contains** "messi"]



Query (publication) language

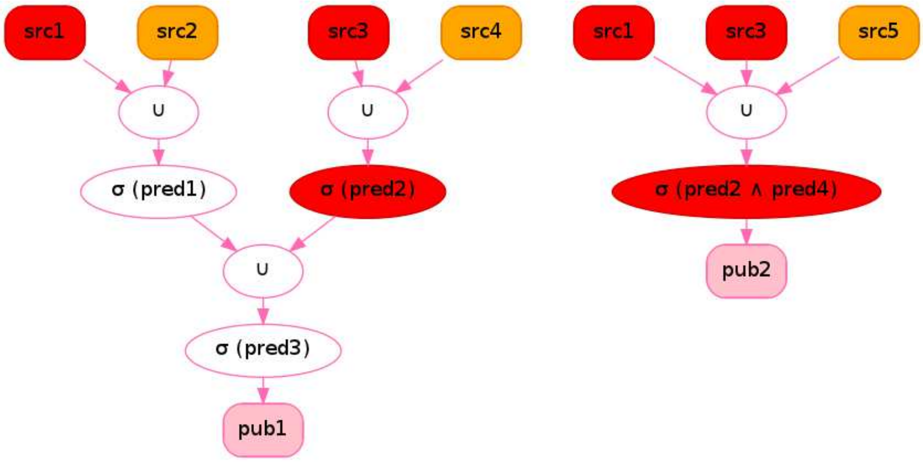
- *Example 3*

create feed myMovies
from allocine **as** $a
join last 3 **weeks on** myFriendsTweets
with $a[title **similar window**.title]
where $a[description **not contains** "julia roberts"]



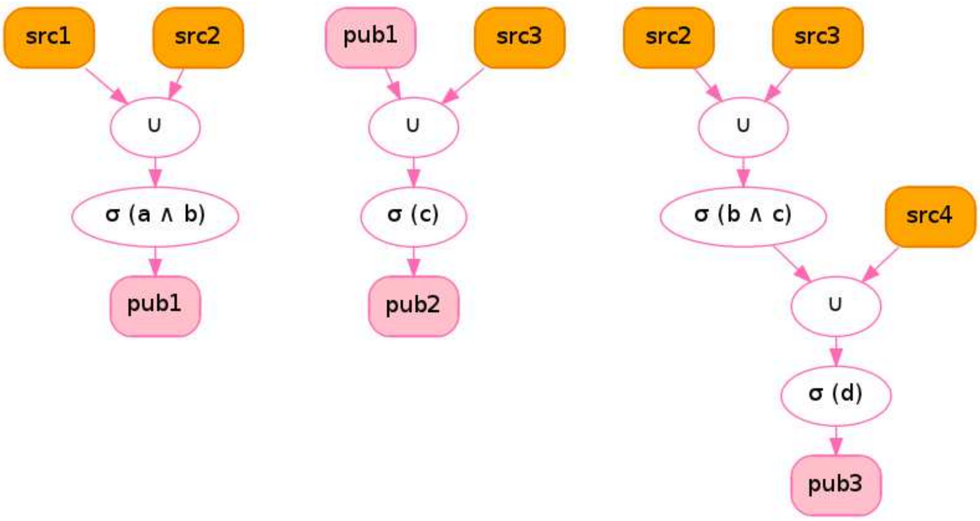
Query optimization

- Observation: users ask often *similar queries*

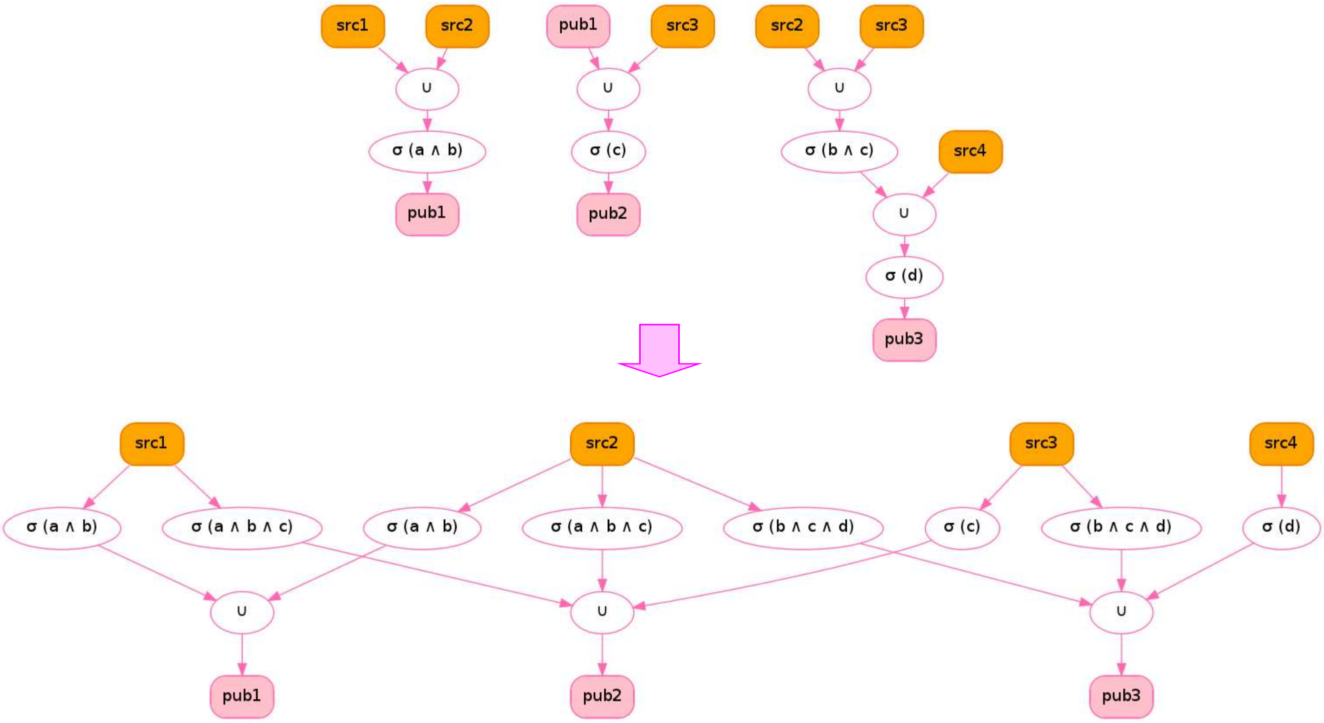


- Goal: factorize operations to reduce the memory space and the processing effort

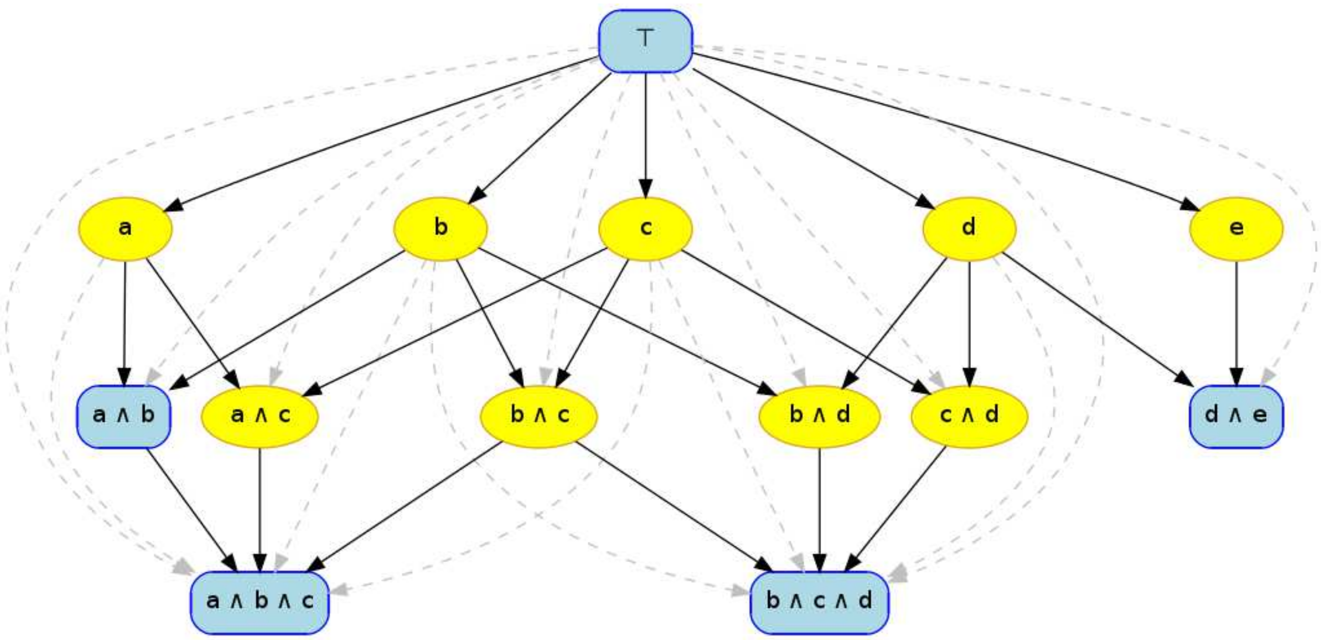
Optimization example



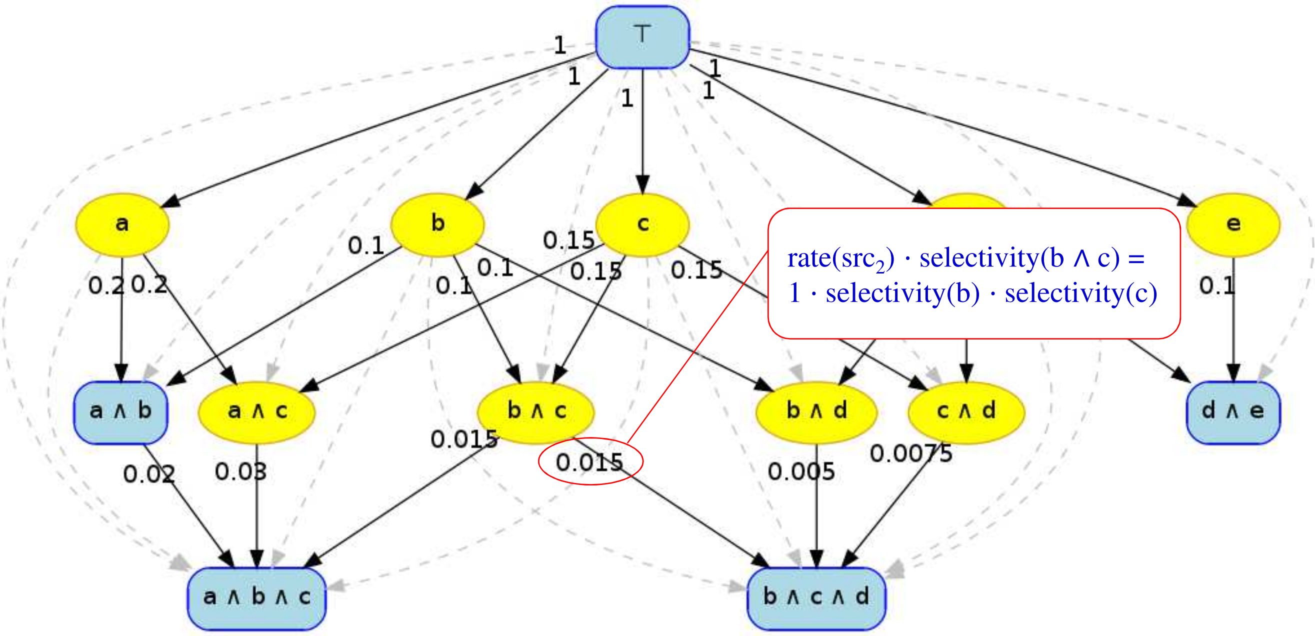
Normalization



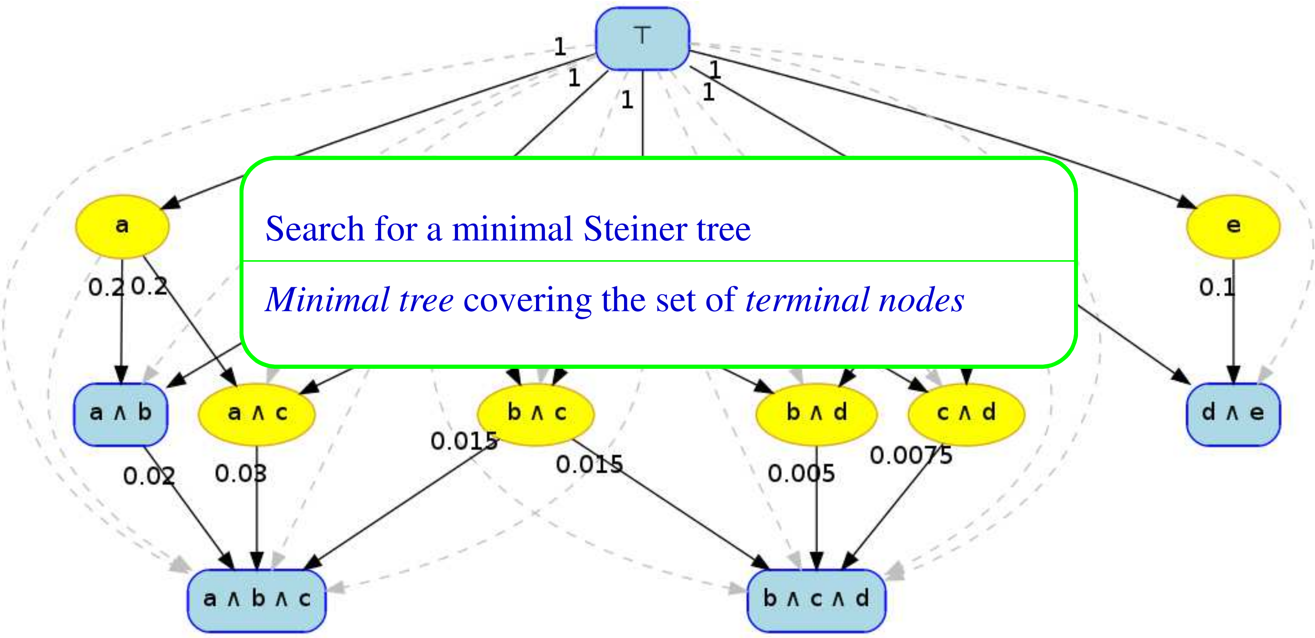
Subsumption graph



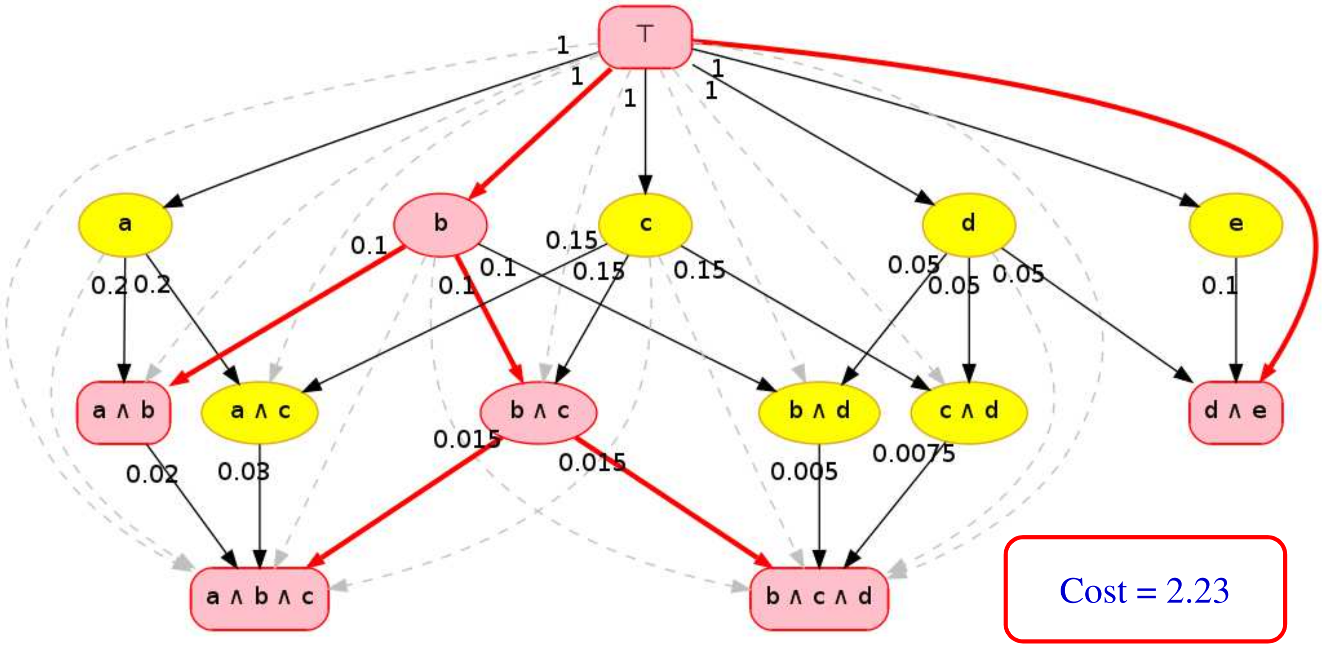
Subsumption graph



Subsumption graph

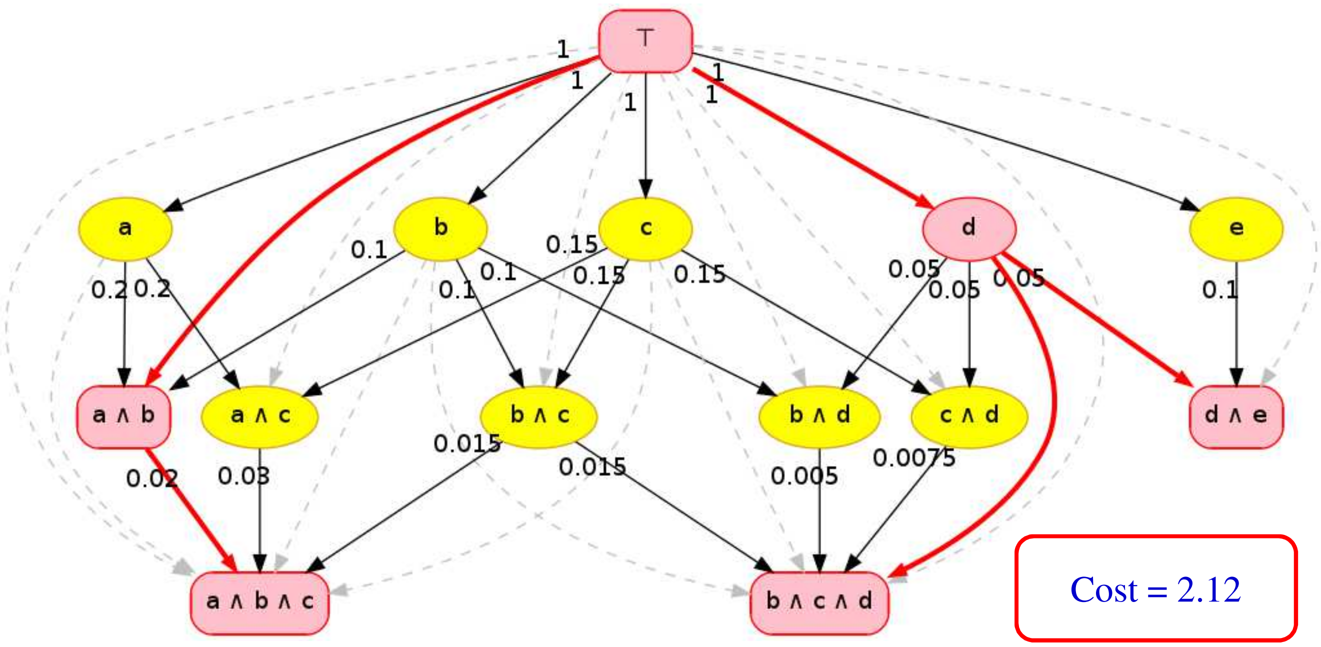


Steiner tree



Cost = 2.23

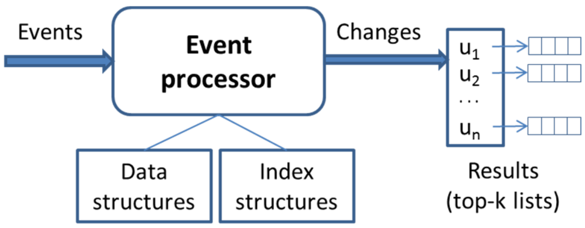
Steiner tree



Cost = 2.12

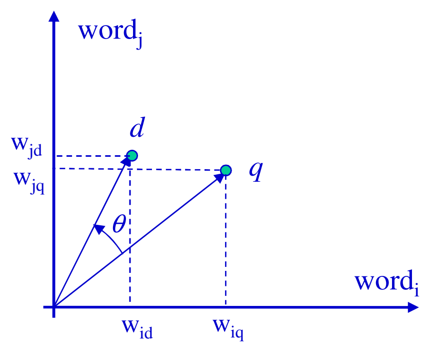
Continuous ranking queries

- Boolean queries
  - An item is relevant or not for a query
  - No ranking among the result items for a query
  - E.g. RoSeS, the example of pub/sub index
- Ranking queries
  - An item has a *relevance degree* for a query
  - Item score for a query = relevance degree → ranking is possible
  - Ranking query types:
    - Items with scores above a given threshold
    - **The best *k* items (top-k)**



Vector model for textual similarity

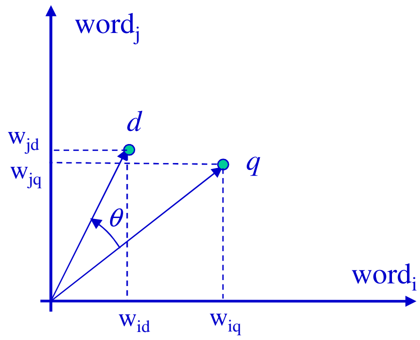
- Vector model
  - Vocabulary of N words
    (e.g. words appearing in the queries)
  - A document *d* (item) = point in the N-dimensional space of the vocabulary
    - Coordinate on the dimension of word *m*: weight w_{md} of word *m* in document *d*
  - Same thing for query *q*



- Textual similarity
  - "Proximity" of vectors representing the documents / queries

Similarity in the vector model

- Generally: cosine of the angle between the two vectors
  - $sim(d,q) = \cos(\theta) = \vec{d} \cdot \vec{q} / (\|\vec{d}\| * \|\vec{q}\|) = \sum_m (w_{md} * w_{mq}) / (\|\vec{d}\| * \|\vec{q}\|)$
  - Normalized vectors:
    $sim(d,q) = \vec{d} \cdot \vec{q} = \sum_m (w_{md} * w_{mq})$



- Computing the weights w_{md}
  - Most common model: ***tf-idf***
  - *Term frequency* (tf): measures the number of occurrences of *m* in *d*
  - *Inverse document frequency* (idf): measures the capacity of the word to differentiate the documents
  - Other parameters may be used (e.g. document size)
    $w_{md} = \alpha_{md} * tf_{md} * idf_m$

Continuous processing of top-k textual queries

- Queries (~documents)
  - Expected result = the *k* best items for each query (top-k)
  - w_{mq} already computed and the queries are indexed
  - Common index used: *inverted file*
    word *m* → sorted list of queries *q* in descending order of w_{mq}

$\mathbf{m_1}$	.....	$\mathbf{m_i}$	.....	$\mathbf{m_n}$
$\mathbf{q_{1,1}}(w_{1,1})$		$\mathbf{q_{i,1}}(w_{i,1})$		$\mathbf{q_{n,1}}(w_{n,1})$
$\mathbf{q_{1,2}}(w_{1,2})$		$\mathbf{q_{i,2}}(w_{i,2})$		$\mathbf{q_{n,2}}(w_{n,2})$
.....		.....		.....

- Arrival of an item *d*
  - Compute w_{md} for all words *m* of *d*
  - For each word *m* of *d* → traverse the index list of *m* → candidate queries for *d* by decreasing degree of interest
  - For each candidate query *q* : evaluate *sim*(*d*, *q*)
    - Possibly *d* may enter the top-k for *q*
  - Various strategies to limit the number of processed candidates

Example of algorithm

- COL-Filter algorithm

  score(d, q) = $\sum_{m \in d} (w_{md} * w_{mq})$

  – For each query q: a list of top-k items and a threshold μ_q (k-th score)

  – Index = lists of q for each word m sorted in descending order of w_{mq} / μ_q

m_1	. . . . .	m_i	. . . . .	m_n
$q_{1,1}(w_{1,1}/\mu_{1,1})$		$q_{i,1}(w_{i,1}/\mu_{i,1})$		$q_{n,1}(w_{n,1}/\mu_{n,1})$
$q_{1,2}(w_{1,2}/\mu_{1,2})$		$q_{i,2}(w_{i,2}/\mu_{i,2})$		$q_{n,2}(w_{n,2}/\mu_{n,2})$
. . . . .		. . . . .		. . . . .

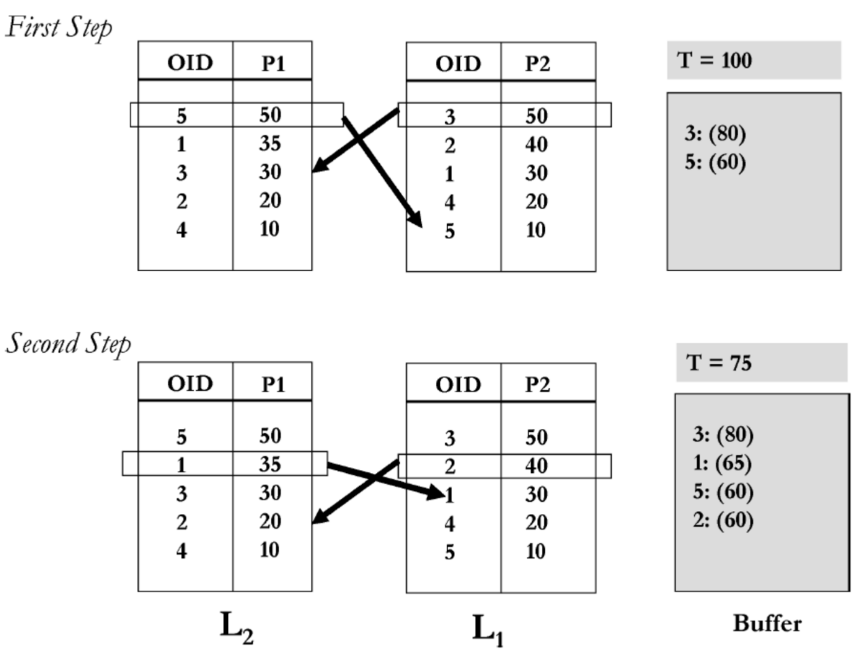
  – $f_d(q)$ = score(d, q) - $\mu_q = \sum_{m \in d} (w_{md} * w_{mq}) - \mu_q$

    - The top-k list of q is updated if $f_d(q) > 0 \rightarrow \sum_{m \in d} (w_{md} * w_{mq} / \mu_q) > 1$

  – Threshold Algorithm

    - Candidate queries considered following the list order: $q_{1,1}$, $q_{2,1}$, …, $q_{n,1}$, then $q_{1,2}$, $q_{2,2}$, …
    - If V_m is the last w_{mq} / μ_q seen in the list of **m** $\rightarrow$ V_m decreases during the traversal
    - If $F_d(q) = \sum_{m \in d} (w_{md} * V_m) \rightarrow F_d(q)$ decreases during the traversal
    - When $F_d(q) \leq 1$ the algorithm can stop

Threshold Algorithm



Source: Ilyas, I. F., Beskales, G., and Soliman, M. A. 2008. A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40, 4, Article 11 (October 2008)

Information streams in social networks

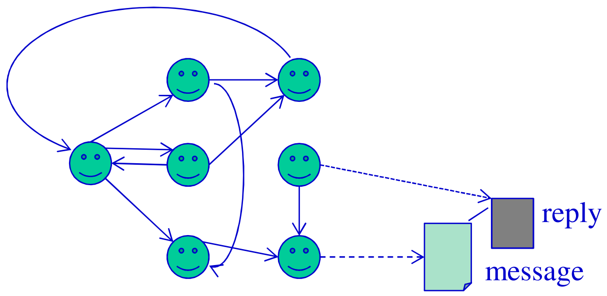
- Classical information streams on the web
  - The user does not have an important role
  - Sources and users (queries) are not related
  - Relevance of an item for a query → textual content criteria
- Social networks
  - The user plays a central role
    - Users produce messages (items)
    - Users consume messages
    - Users may interact with messages (like, comment, …)
    - Relations between users
  - The relevance of a message for a query → textual + social criteria
  - Message importance decreases in time

Types of social networks

- Entities in a social network
  - *Users*: with possibly explicit links between them
  - *Content*: documents (web pages, photos, videos, etc.)
    - May also have links between them (web pages)
  - *Messages*: text + possibly links to documents
    - Sometime: the message may be a simple tag associated to a document
- Three main types of social networks
  - Unidirectional networks (Twitter)
  - Symmetric networks (Facebook)
  - Tagging networks (Flickr)
- In practice: a mix of different types
  - Facebook: also unidirectional for the fan pages
  - Flickr: tags and friendship links (for access control to published content)

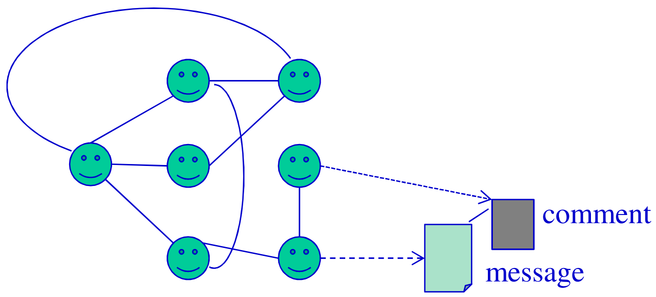
Unidirectional networks

- A user can follow the messages of other users
  - E.g. Twitter
  - Public text messages
  - Documents indirectly addressed through links in the text
  - Hashtags, localization, timestamp, …
  - Interaction with messages: re-tweet, reply, favorite
- Implicit query = the messages of the followed users
  - Other queries: explicit – by hashtag, by keywords



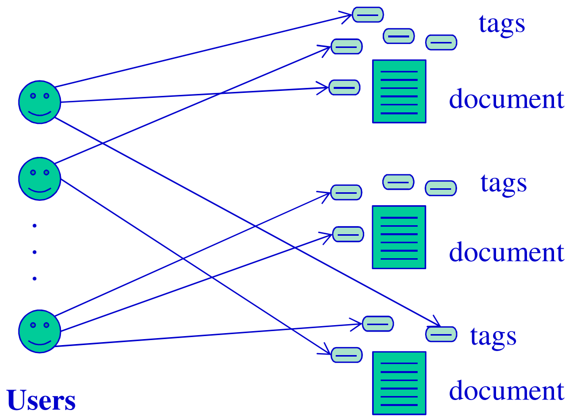
Symmetric networks

- Friendship links between users (symmetric)
  - E.g. Facebook
  - Private text messages, visible by the friends
  - Documents indirectly addressed through links in the text
  - Interaction: like, comment, …
- Implicit query = the messages of the friends



Tagging networks

- Users associate tags to documents
  - Unconstrained tags (“folksonomy”) or predefined tags
  - No explicit link between users
  - E.g. Delicious (bookmarks), Flickr (photos)
- Tags associated to a document → descriptive meta-document
  - Search by tags = textual search on the descriptive meta-documents
- Implicit link between users
  - u_1 and u_2 use similar tags for the same documents
  - u_1 tags a document produced by u_2



Information streams in social networks

- Messages produced by the users
  - Stream of textual messages, of tags, interactions
- Queries: various forms of textual monitoring queries
  - Generalization:
    - User profile defined by *a set of terms* (weighted)
    - Implicit textual query based on these terms, on all the followed streams
  - Relevance: textual content + social network criteria
  - Message importance decreases in time

Example of relevance model in a social network

- Importance for user *u* of a message *m* published by u^m

score (m, u) = α content (m, u) + (1-α) social (m, u)

content (m, u) = *similarity* (m, profile(u))

social (m, u) = β ***global*** (m) + (1 - β) ***local*** (u, u^m)

global (m) = γ *importance* (u^m) + (1 - γ) *interaction* (m)
local (u, u^m) = *relative-importance* (u, u^m)

- Score criteria
  - Content score: content similarity between message and user profile
  - Global social score: emitter importance, interaction with the message
  - Local social score: relative importance of the emitter for the user in the social network

Considering time

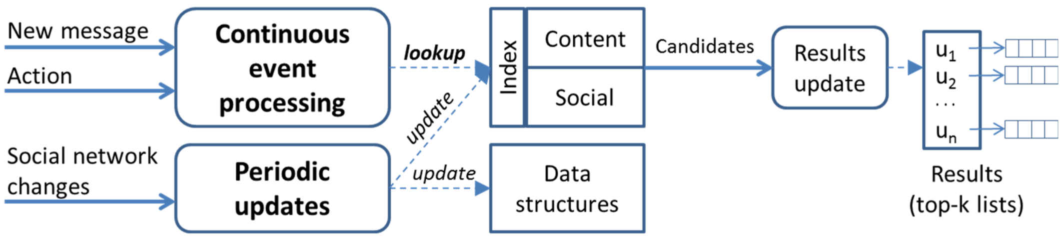
- Decrease of the importance of messages in time
- Two main approaches proposed so far
  - Limited period of interest: *sliding temporal window*
    - Message deleted when exiting the window
  - Continuous decrease of the score: *decay function* **TD(Δt)**
    - TD : $R_+ \rightarrow [0, 1]$ decreasing, with TD(0) = 1
    - For a message *m* published at t_m : **tscore(m,u,t) = score(m,u) TD(t-t^m)**
    - *Order preserving decay function*:
      if **tscore(m_1,u_1,t) ≤ tscore(m_2,u_2,t)** then
      tscore(m_1,u_1,t') ≤ tscore(m_2,u_2,t'), ∀t'>t
  - In practice: *bonus function* **TB(Δt)** relative to a time origin t_0
    - Advantage: not changing in time

$$\textbf{\textit{tscore}}(m, u, t) = \textbf{\textit{score}}(m, u) \cdot \textbf{TB}(t^m - t_0)$$

$$\textbf{TB} : \mathbf{R}_+ \rightarrow [1, \infty) \text{ monotonically increasing}$$

Ranking queries for social networks

- More difficult compared to classical web information streams
  - More complex relevance function (textual + social)
  - Management of the time factor
  - Considering interactions with messages
  - Top-k update on several event types
- Events to consider
  - New published message
  - New interaction with a message
  - Creation/deletion of links between users



Example: the SANTA algorithm

- Scoring function
  - Content-based scoring: normalized cosine similarity
  - Local social scoring: relative importance function $f(u_i, u_j)$
  - Global social scoring: $G(m)$

$$score(m,u) = a \sum_{t_i \in m} w_{im} w_{iu} + b\, f(u, u^m) + c\, G(m)$$

  - Update condition: message m enters the top-k list of user u
    - μ_u = *k-th score of the user u query*

$$F(m,u) = score(m,u) - \mu_u > 0$$

$$a \sum_{t_i \in m} w_{im} \boxed{w_{iu}} + b \boxed{f(u, u^m)} + c\, G(m) + \boxed{-\mu_u} > 0$$

The SANTA index structure

- Simple and extensible index structure
  - Efficient, easily parallelizable
- Minimizes the update effort
- Threshold algorithm given the monotonicity of F(m, u)