Personalised Graph-based Selection of Web APIs

Milan Dojchinovski[1], Jaroslav Kuchar[1], Tomas Vitvar[1] and Maciej Zaremba[2]

[1] Web Engineering Group
Faculty of Information Technology
Czech Technical University in Prague
firstname.lastname@fit.cvut.cz
[2] Digital Enterprise Research Institute
National University of Ireland, Galway
maciej.zaremba@deri.org

Abstract. Modelling and understanding various contexts of users is important to enable personalised selection of Web APIs in directories such as Programmable Web. Currently, relationships between users and Web APIs are not clearly understood and utilized by existing selection approaches. In this paper, we present a semantic model of a Web API directory graph that captures relationships such as Web APIs, mashups, developers, and categories. We describe a novel configurable graph-based method for selection of Web APIs with personalised and temporal aspects. The method allows users to get more control over their preferences and recommended Web APIs while they can exploit information about their social links and preferences. We evaluate the method on a real-world dataset from ProgrammableWeb.com, and show that it provides more contextualised results than currently available popularity-based rankings.

Keywords: Web APIs, Web services, personalisation, ranking, service selection, social network

1 Introduction

The rapid growth of Web APIs and a popularity of service-centric architectures promote a Web API as a core feature of any Web application. According to ProgrammableWeb[3], a leading service and mashup directory, the number of Web APIs has steadily increased since 2008. While it took eight years to reach 1,000 APIs in 2008, and two years to reach 3,000 in 2010, it took only 10 months to reach 5,000 by the end of 2011 [16]. In spite of this increase, several problems are starting to arise. Old and new not yet popular Web APIs usually suffer from the preferential attachment problem [14], developers can only run a keyword-based search in a service directory or they run a Google search to find Web pages that reference or describe Web APIs. Although there exist a number of sophisticated mechanisms for service discovery, selection and ranking, there is still a lack of

[3] http://www.programmableweb.com

methods that would in particular take into account a wider Web APIs' and developers' contexts including developers' profiles, information who developed Web APIs or used them in a mashup, Web APIs' or mashups' categories as well as the time when an API or a mashup was developed or published. With the popularity of Web APIs and directory services like ProgrammableWeb, it is now possible to utilize all such information in more sophisticated service selection methods.

In this paper we develop a novel Web API selection method that provides personalized recommendations. As an underlying dataset we create so called *Linked Web APIs*, an RDF representation of the data from the ProgrammableWeb directory, that utilizes several well-known RDF vocabularies. The method has the following characteristics: 1) *social and linked*–it exploits relationships among Web APIs, mashups, categories, and social relationships among developers such as who knows who in the ProgrammableWeb directory, 2) *personalized*–it takes into account user's preferences such as developers the user knows and preferences that define importances of predicates, and 3) *temporal*–it takes into account a time when Web APIs and mashups appeared in the graph for the first time.

We develop a method called the *Maximum Activation* and show how it can be used for the Web API selection. The method calculates a maximum activation from initial nodes of the graph (defined by a user profile), to each node from a set where a node in the set represents a Web API candidate. We adopt the term *activation* from the spreading activation method[1] and we use it as a measure of a connectivity between source nodes (initial nodes defined by a user profile) and a target node (a Web API candidate). We use flow networks as an underlying concept for evaluation of the maximum activation in the graph. We implement the method as a Gephi plugin,[4] and we evaluate it on several experiments showing that the method gives better results over traditional popularity-based recommendations.

The remainder of this paper is structured as follows. Section 2 describes the underlying Linked Web APIs dataset and Section 3 describes the maximum activation method, its definitions and the algorithm. Section 4 describes several experiments from running the method on the Linked Web APIs dataset and a case study that shows how a developer can use the method when creating a mashup with various Web APIs. Section 5 describes the related work that also includes information on how the method compares to the spreading activation method. Finally, Section 6 concludes the paper and describes our future work.

2 Linked Web APIs

Figure 1 shows an extract of the Linked Web APIs dataset of the ProgrammableWeb, currently the largest directory of Web APIs. The Linked Web APIs dataset represents the whole directory as an RDF graph with over 300K RDF triples. To build the dataset we gathered data about Web APIs, mashups, their categories,

[4] https://github.com/jaroslav-kuchar/Maximum-Activation-Method

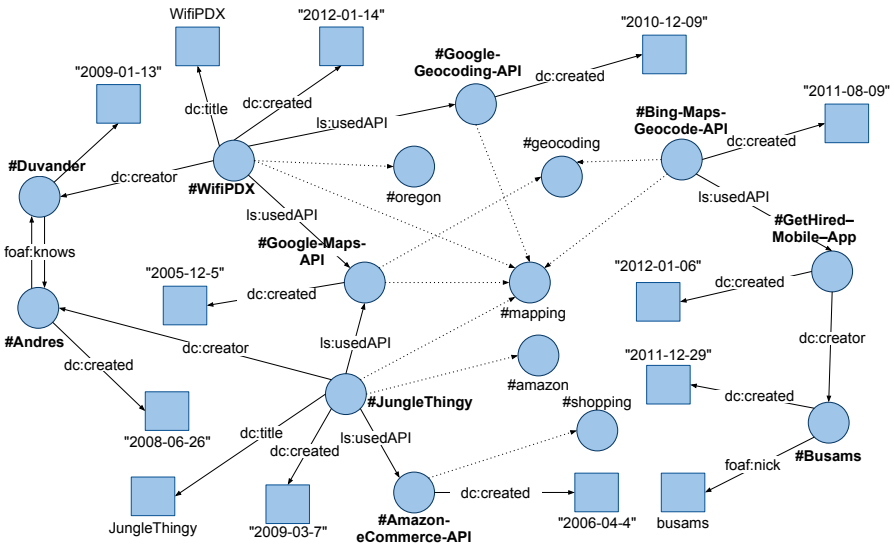


Fig. 1. Excerpt from the Linked Web APIs dataset

developer profiles, and relationships among Web APIs and mashups that use them, relationships among Web APIs, mashups and developers who developed them, and relationships among Web APIs, mashups and categories. Moreover, we also capture the time of the Web APIs and mashups when they appeared in the ProgrammableWeb directory for the first time.

Note that there are other information in the ProgrammableWeb that we could use to make the Linked Web APIs richer such as various technical information about protocols and data formats. Also, we could better associate the data with other datasets in the Link Data cloud and publish it to the Linked Data community. Although we plan to do this in our future work, the Linked Web APIs dataset that we present here already provides the sufficient information for our Web API selection method.

The Linked Web API dataset uses several well-known ontologies. Concepts from FOAF[5] ontology (prefix `foaf`) represent mashup developers as *foaf:person* concepts with their social links, concepts from the WSMO-lite [15] ontology (prefix *wl*) represent Web APIs as *wl:service* concepts and their functional category descriptions. We also use the Dublin Core[6] vocabulary (prefix `dc`) for properties such as *title*, *creator* and *date*, and the SAWSDL[12] property *sawsdl:modelReference*. Further, we create new concepts and properties for which we use the `ls` prefix. We define the *ls:mashup* concept that represents a mashup and the *ls:category* concept that represents a functional Web API/mashup category. There are following types of edges in the Linked Web APIs:

[5] http://xmlns.com/foaf/0.1/
[6] http://dublincore.org/documents/dcmi-terms/

1. *User—User*: an edge between two user nodes represented with the *foaf:knows* property indicating a social link.
2. *User—Mashup*: an edge between a user and a mashup represented with the *dc:creator* property.
3. *Mashup—API*: an edge between a mashup and an API represented with the *ls:usedAPI* property.
4. *Mashup—Category*, and *API—Category*: an edge between a mashup/API and a category represented with the *sawsdl:modelReference* property.

3 Maximum Activation Method

3.1 Definitions

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{I})$ be a graph representing Linked Web APIs where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges and $\mathcal{I} : \mathcal{E} \rightarrow \mathbb{N}$ is a capacity function which associates a capacity of an edge with a natural number. A node in $\mathcal{V}$ can represent an API described by the *wl:Service* concept, a mashup described by the *ls:Mashup* concept, a user described by the *foaf:Person* concept or a category described by the *ls:Category* concept. An edge $e \in \mathcal{E}$ represents a mutual (bidirectional) relationship between two nodes as follows: for a property in the Linked Web APIs dataset we create an *inverse property* such that when (o_1, p, o_2) is a triple where o_1, o_2 correspond to nodes in $\mathcal{V}$ and p corresponds to an edge in $\mathcal{E}$, we create a new triple (o_1, p^{-1}, o_2) where p^{-1} is an inverse property to p. See Section 3.3 for additional details.

Let $P = \{p_1, p_2, ..., p_n\}$, $p_i \in \mathcal{V}$ be a set of nodes that represent a user profile. The nodes in P may represent the user himself, nodes that the user likes or knows or has any other explicit or implicit relationships with. Further, let $W = \{w_1, w_2, ..., w_m\}$, $w_i \in \mathcal{V}$ be a set of nodes that represent a user request as Web APIs candidates. The Maximum Activation method then calculates a maximum activation a_i for each Web API candidate $w_i \in \mathcal{W}$. The higher number of the maximum activation denotes a Web API candidate with a higher rank, that is the preferred Web API candidate over a Web API candidate with a lower maximum activation.

We denote an activation that can be sent between two nodes linked with an edge e as a natural number $i(e) \in \mathbb{N}$. The activation sent through an edge cannot exceed the capacity of the edge defined by the capacity function

$$\mathcal{I}(e_{i,t}) = S(e_i) * \mathcal{A}(e_{i,t}) \tag{1}$$

where $S(e_i) \in \{x \in \mathbb{N} | 0 \leq x \leq 100\}$ is a user preference function that defines an importance of the edge e_i (i.e., how the user sees an importance of semantics represented by the edge) and $A(e_{i,t})$ is the *exponential ageing function*. An importance $S(e_i) < 50$ indicates that the user does not prefer the edge's semantics, an importance $S(e_i) > 50$ indicates the the user prefers the edge's semantics and the importance $S(e_i) = 50$ indicates a neutral value. A user may chose an

arbitrary number of edges for which he/she defines preferences. Edges for which the user does not define any preferences have a default preference 50.

Further, we define the exponential ageing function as

$$\mathcal{A}(e_{i,t}) = \mathcal{A}(e_{i,t_o}) * e^{-\lambda t} \tag{2}$$

where $\mathcal{A}(e_{i,t})$ is an age of the edge e_i at time t, $\mathcal{A}(e_{i,t_o})$ is the initial age of the edge e_i at the time the edge appeared in the graph $\mathcal{G}$ (i.e., values of *dc:created* property) and λ is an *ageing constant*. The ageing constant allows to configure an acceleration of the ageing process. Since our method gives better results for better connected nodes in the graph, the ageing function allows to control an advantage of "older" nodes that are likely to have more links when compared to "'younger" ones (see Section 4.1 for discussions on how we setup the ageing constant and Section 4.3 and 4.4 for differences in results with and without the ageing function applied).

Note that we currently only apply the ageing function to edges that are linked with nodes representing Web APIs and mashups. In other words, we use a creation date of a Web API or a mashup to evaluate the ageing function of any edge that links with the Web API or the mashup respectively. We assume that the Web API or the mashup was created at the same time along with all its edges that connect it to other nodes in the graph. For all other edges it holds that $\mathcal{A}(e_{i,t}) = 1$.

3.2 Algorithm

We calculate the Maximum Activation according to the following algorithm.

Inputs:

- Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{I})$ constructed from the Web Linked APIs dataset.
- A user profile $P = \{p_1, p_2, ..., p_n\}$.
- Web API candidates $W = \{w_1, w_2, ..., w_m\}$.
- A user preference function $S(e_i)$.

Output:

- A set of maximum activations $\{a_i\}$ evaluated for each $w_i \in W$.

Uses:

- A set $C = \{e_1, e_2, ..., e_k\}$, $e_i \in \mathcal{E}$.
- A function FF that represents the Ford-Fulkerson algorithm [8].

Algorithm:

1: *// create a virtual source node p'*
2: add node p' to $\mathcal{V}$
3: **for all** $p_i \in P$ **do**
4: add edge $e(p', p_i)$ to $\mathcal{E}$, $S(e) \leftarrow 100000$, $\mathcal{A}(e) \leftarrow 1$
5: **end for**
6: *// calculate a maximum activation a_i from*

7: *// a virtual node p' to every Web API candidate w_i*
8: **for all** $w_i \in W$ **do**
9: $C \leftarrow FF(p', w_i, \mathcal{G})$
10: $a_i \leftarrow \sum_{e_i \in C}(\mathcal{I}(e_i))$
11: **end for**

In lines 2–5, the algorithm first creates a virtual node representing a single source node with links connecting the virtual node and all other nodes from the user profile. Any edge that connects the virtual node with any other node in the graph has a capacity set to a very large value so that the edge does not constrain the maximum activation. In lines 8–11, the algorithm finds a maximum activation for each Web API candidate w_i from the virtual node p'. For this we use the Ford-Fulkerson algorithm to find a maximum activation from the *source node* (i.e., the virtual node) to the *target node* (i.e., a Web API candidate). We do not formaly describe the *FF* algorithm here, however, for the purposes of later discussion in Section 3.3 we provide its brief description: the *FF* algorithm first sets the initial activation for each edge to 0 and tries to find an *improving path* on which it is possible to increase the activation by a minimum value greater then 0. If such path is found, the algorithm increases activations on every edge on the path and tries to find another path. When no more path is found, the algorithm ends. The result of *FF* is the set C that contains every last edge from all paths from the source towards the target when an improving path is not possible to find. The maximum activation is the sum of all activations on edges from C. In line 10, the algorithm finally calculates the maximum activation as a sum of all activations of edges in C.

3.3 Discussion

Meaning of Maximum Activation Value. As we noted earlier, we interpret the maximum activation of the graph as a measure that indicates how well the source nodes are connected with the target. In general, the more improving paths exist between the source and the target, the higher maximum activation we can get. However, the value of the maximum activation is also dependent on constraints and the creation time of Web APIs and mashups along the improving paths when the ageing function is applied.

Maximum Activation and Edges in C. The edges in C are constraining the maximum activation which means that if capacities of such edges increase, the maximum activation can be increased. Note, however, that we assign capacities based on semantics of egdes thus by changing a capacity on an edge in C, we also change capacities on other edges not in C. Running the algorithm again on the graph with new capacities will lead to a different set C and different maximum activation. In other words, it does not hold that increasing a capacity on any edge in C will lead to a higher maximum activation. This also means that maximum activation that our algorithm evaluates has a global meaning while activations on individual edges do not have any meaning. Defining capacities for individual edges is the subject of our future work.

Graph $\mathcal{G}$ Construction. When we construct the graph $\mathcal{G}$ from the Linked Web APIs dataset, for every predicate we create a bidirectional edge. A graph with bidirectional edges provides a richer dataset for maximum activation evaluation. A large graph with unidirectional edges may contain many dead end paths that may limit the number of improving paths that the algorithm would be able to find from the source to the target nodes. Evaluation of maximum activation on such graph would not provide many interesting results.

4 Experiments

In this section we present several experiments and their results[7] that use maximum activation for the Web APIs selection.

For our experiments we use the full Linked Web APIs dataset. The dataset contains all user profiles for users that created at least one mashup. We also extracted profiles on all categories, tags, mashups and Web APIs. The snapshot we use covers the period from the first published API description in June 2005, till May 18th, 2012. The snapshot includes 5 988 APIs, 6 628 mashups and 2 335 user profiles. In the experiments we addressed following questions:

- *What is the impact of user preference function on results of the maximum activation?*
- *How does the ageing factor influence the maximum activation?*
- *How can the popularity of an API evolve over time?*
- *How to make the process of building a mashup more personalised and contextualised?*

4.1 Setting the Ageing Constant

We experimentally set the ageing constant to a value $\lambda = 0.1$ and the age period to one week ($t = week$). Our graph contains data since June 2005, that is approximately 360 weeks. Figure 2 depicts an effect on ageing function for different λ. Note that the higher the constant is, the algorithm promotes the more recently added APIs and Mashups.

4.2 Impact of the User Preference Function

The user preference function defines an importance of the edge, that is how the user sees the importance of semantics represented by the edge. For example, the user can give a higher importance to edges representing a friendship (*foaf:knows*) than to edges between mashups and Web APIs (*ls:usedAPI*). The importance values, along with the chosen edge ageing constant λ, are used to compute the total capacity of an edge (see definition (1)). To study the influence of an importance value on a single edge, we were gradually increasing the value from 0

[7] Full results of the experiments are available at `http://goo.gl/GKIbo`

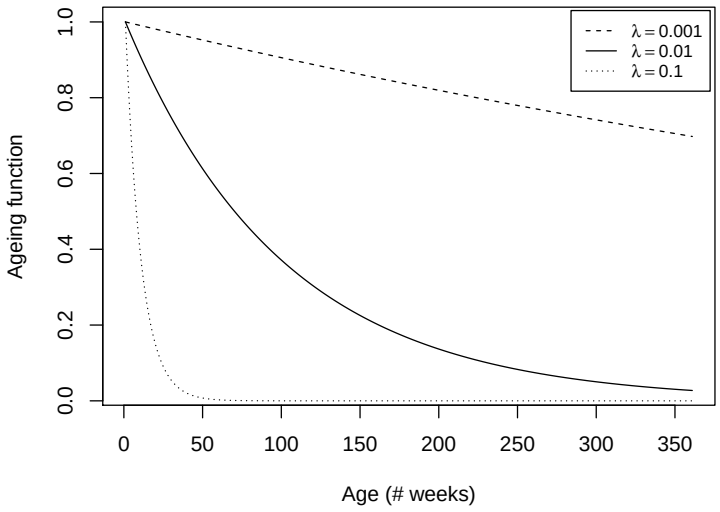


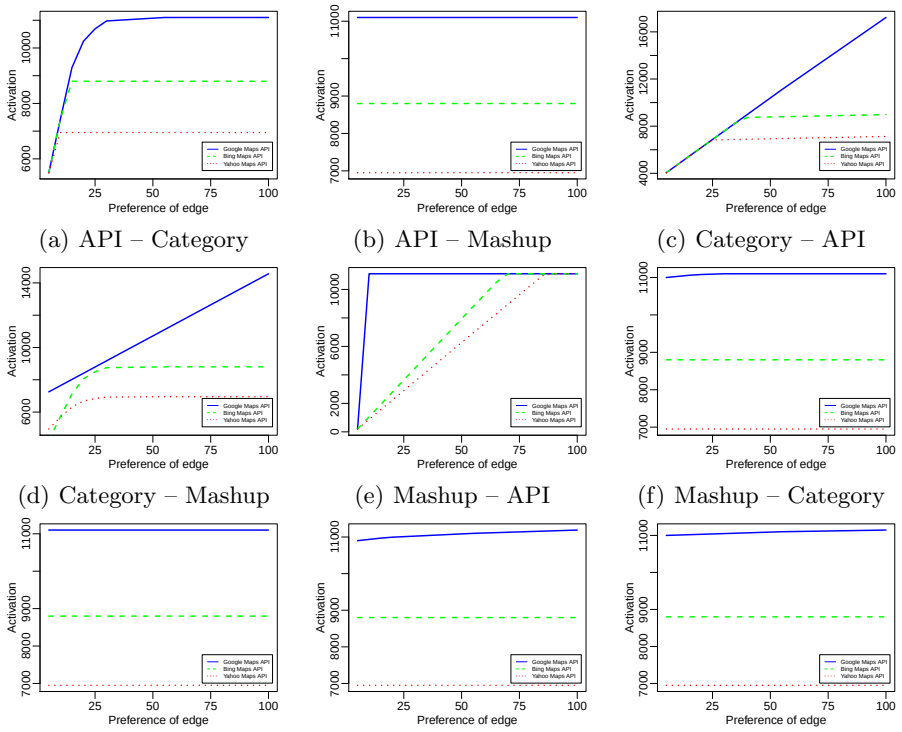
Fig. 2. Ageing function

to 100 by a step of 5 and fixed importance values of all other edges to 50. We run this experiment for 3 different well-known APIs, namely Google Maps, Bing Maps and Yahoo Maps.

Figure 3 shows the experiment results: the importance value on the edge *API–Mashup* (Fig.3(b)) and *Mashup–User* (Fig.3(g)) does not have influence on the maximum activation. Slight influence has the importance value on edges *Mashup–Category* (Fig.3(f)), *User–Mashup* (Fig.3(h)) and *User–User* (Fig.3(i)). Fig.3(a) further shows that different importance values have various ranges of influence: the importance value for the *API–Category* has influence in a range $0-5$ for Yahoo Maps API, $0-10$ for Bing Maps API, and $0-15$ for Google Maps API, while higher importance values do not have any influence as the maximum activation is limited by the capacities of other types of edges.

4.3 Impact of the Ageing Constant λ

The ageing constant λ is a configurable parameter which influences the value of assigned edge capacity. The higher the λ is, the more recent edges will be preferred – that is, the older edges will have a lower capacity. In different datasets edges can occur more or less frequently therefore appropriate value for the λ should be set. Setting high λ in datasets where the edges occur less frequently may lead to very low edge capacities and consequently to the low activation value. In other words, the ageing constant λ makes the selection method more adaptable to different datasets.



(a) API – Category (b) API – Mashup (c) Category – API

(d) Category – Mashup (e) Mashup – API (f) Mashup – Category

(g) Mashup – User (h) User – Mashup (i) User – User

Fig. 3. Impact of Importance values

For this experiment we chose a random user Dave Schappell[8] and we calculated the maximum activation for each API candidate in the "mapping" category. We evaluated the results in the period from 1st of June 2009 (shortly after the user registered his profile) till 1st of June 2012 with a period of age set to 1 week. We set the ageing constant λ to values 0.01 and 0.1. By setting the ageing constant we are able to accelerate the ageing process, that is we get a lower capacity on older edges. Fig.2 shows, setting the ageing constant to 0.1 we get higher maximum activation for edges that appeared in the last 50 weeks, and setting it to 0.01 in the last 350 weeks.

Table 1 shows the configuration of importance values for various types of edges for this experiment and Table 2 and 3 shows the results of this experiment for λ set to 0.01 and 0.1 respectively. In these tables, the"PW rank" column shows a popularity-based ranking used by the ProgrammableWeb which is only based on a number of mashups used by an API. Google Maps API is the highest ranked API by our method (for both λ=0.01 and λ=0.1) and also is the highest ranked by the Programmable Web popularity-based method. For $\lambda = 0.01$, the

[8] http://www.programmableweb.com/profile/daveschappell

Table 1. Importance Value Configuration

Edge name	Importance value	Edge name	Importance value
API–Category	50	Mashup–Category	70
API–Mashup	50	Mashup–User	50
Category–API	70	User–Mashup	90
Category–Mashup	20	User–User	90
Mashup–API	70	/	/

Table 2. Summarised ranking results with λ=0.01

Node ID	API name	Date created	Max-Activation $\lambda = 0.01$		PW rank
			value	rank	
2053	Google Maps API	2005-12-05	**5559**	1	1
2041	Google Earth API	2008-06-01	1080	2	5
2057	Google Maps Data API	2009-05-20	1043	3	8
2052	Google Geocoding API	2010-12-09	1028	4	11
3032	Microsoft Bing Maps API	2009-06-09	853	5	2
2060	Google Maps Flash API	2008-05-17	792	6	6
5827	Yahoo Geocoding API	2006-02-14	715	7	4
5836	Yahoo Maps API	2005-11-19	707	8	3
493	Bing Maps API	2009-06-09	662	9	10
2070	Google Places API	2010-05-20	553	10	18

Table 3. Summarised ranking results with λ=0.1

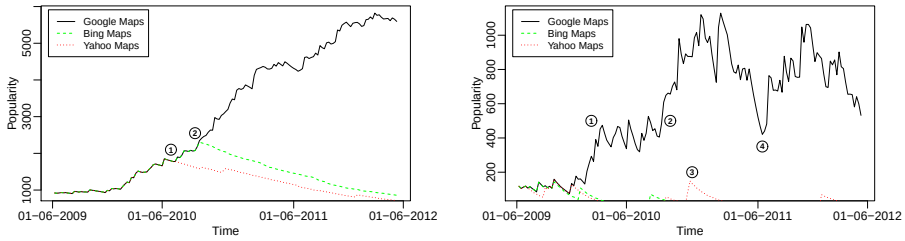
Node ID	API name	Date created	Max-Activation $\lambda = 0.1$		PW rank
			value	rank	
2053	Google Maps API	2005-12-05	**503**	1	1
5531	Waytag API	2012-04-27	210	2	230
4330	Scout for Apps API	2012-04-20	190	3	202
4535	Google Geocoding API	2010-12-09	184	4	11
3815	Pin Drop API	2012-03-27	135	5	191
5950	Zippopotamus API	2012-04-26	123	6	233
5825	Yahoo Geo Location API	2012-04-23	120	7	230
1836	FreeGeoIP API	2012-03-29	112	8	116
5156	Trillium Global Locator API	2012-04-18	111	9	109
1430	eCoComa Geo API	2012-05-15	108	10	108

method favors the recent APIs but also does not ignore APIs that were actively used in the past 350 months (approx. 7 years).

From the results in Table 3 it is possible to see that the ageing constant $\lambda = 0.1$ promotes newer APIs while at the same time it does not ignore the all-time popular APIs such as Google Maps API and Google Geocoding.

4.4 Popularity of APIs over Time

In this experiment we examine a popularity of 3 APIs from the "mapping" category for the user Dave Schappell in different points in time. We use the configuration in Table 1 and the ageing constant λ set to values 0.01 and 0.1.



(a) With ageing constant $\lambda = 0.01$ (b) With ageing constant $\lambda = 0.1$

Fig. 4. API popularity over Time

The results show that the Google Maps API has the highest popularity in both cases for the ageing constant set at 0.01 and 0.1. From Figure 4(a) we can see that the popularity of Yahoo Maps API and Bing Maps API follows the popularity of the Google Maps API until the time marked with (1) and (2). After the times (1) and (2), a popularity of the two APIs starts to fall. Around December 2010 and January 2012 the popularity of Yahoo Maps API experienced minimal activation growth due to several new mashups that were created and used this API.

Figure 4(b) shows a popularity of the three APIs with a more strict edge ageing. After the first half year, when the popularity of the 3 APIs is nearly the same, the popularity of the Google Maps API is starting to increase until the time marked with (1) and stays at this level until the time marked with (2). Between the times (2) and (4) Google Maps API gained a popularity up to maximum activation of 1 129, however, it also started to lose some activation due to a less number of mashups that were using this API. On the other hand, popularity of the Yahoo Maps API increased around December 2010 (3) due to its more intensive usage. As we can see, in certain cases, by using the ageing function we can get better results than the PW'S popularity-based ranking.

4.5 Case Study

In this section we present a case study for personalised API selection to illustrate capabilities of our maximum activation method. We have a developer who wants to improve tourists' experience in New York, USA by creating the Visitor Mashup. The Visitor Mashup should aggregate information about different events and information about restaurants in the city and in the area of New

York. Information about various points of events and restaurants should be layered on the map and dynamically updated when tourists change their locations and new events and restaurants become available.

Developer starts the process of building the Visitor Mashup by identifying groups of relevant APIs. As he progresses and selects APIs, the ranking process becomes more personalised and contextualised. The process of creating the Visitor Mashup is described by following steps when in each step the developer selects one API:

– **Maps API.** Developer builds his profile adding "maps" and "location" categories to it. He assigns a high importance value to the "API–Category". Table 4 shows the highest ranked results: Google Maps, Microsoft Bing Maps and Yahoo Maps. The developer decides to select the Google Maps API.

Table 4. Summarised ranking results for Maps API

Node ID	API name	Date created	Max-Activation λ not set		Max-Activation $\lambda = 0.01$		PW rank
			value	rank	value	rank	
2053	Google Maps API	2005-12-05	**13720**	1	**6509**	1	1
3032	Bing Maps API	2009-06-09	3720	2	238	2	10
5836	Yahoo Maps API	2005-11-19	2980	3	172	3	3

– **Events API.** The developer further searches for events API by updating his profile with "events" category, adding "Google Maps API" and preserving "maps" and "location" categories. Further, he increases an importance value of the "Mashup–API". Table 5 shows highest ranked results: Seatwave, Eventful and Upcoming.rg. The developer selects Seatwave API.

– **Restaurant API.** The developer searches restaurants API by adding "'food", "restaurants" and "menus" categories to his profile. This time the developer decides to use his social links and to look for APIs used by his friends developers that he adds to his profile. Table 6 shows the highest ranked APIs SinglePlatform, Menu Mania and BooRah. The developer selects SinglePlatform API for restaurant information and recommendations.

Table 5. Summarised ranking results for Events API

Node ID	API name	Date created	Max-Activation λ not set		Max-Activation $\lambda = 0.01$		PW rank
			value	rank	value	rank	
4348	Seatwave API	2012-02-28	940	3	**842**	**1**	4
1578	Eventful API	2005-10-31	3930	1	710	2	1
5371	Upcoming.rg API	2005-11-19	3220	2	411	3	2

Table 6. Summarised ranking results for Restaurant API

Node ID	API name	Date created	Max-Activation λ not set		Max-Activation $\lambda = 0.01$		PW rank
			value	rank	value	rank	
4522	SinglePlatform API	2012-01-30	150	2	**125**	1	6
2980	Menu Mania API	2009-12-05	**220**	1	65	2	1
611	BooRah API	2008-10-31	120	3	30	3	3

5 Related Work

Graph-based representation of services is a relatively new approach. The authors in [2] propose service selection based on previously captured user preferences using the "Follow the Leader" model. In [14] the authors construct collaboration network of APIs and propose a social API Rank based on the past APIs' utilisations. Other approaches that rank services based on results from social network-based analyses in social API networks can be found in [17] and [13].

A particular method that relates to our work is the already mentioned spreading activation. It is a graph-based technique, originally proposed as a model of the way how associative reasoning works in the human mind [4]. The spreading activation requires directed semantic network, e.g. an RDF graph [5,9,7]. The inputs of the basic spreading activation algorithm are number of nodes with an initial activation which represent a query or interests of a user. In sequence of iterations initial (active) nodes pass some activation to connected nodes, usually with some weighting of connections determining how much spread gets to each. This is then iterated until some termination condition is met. The termination conditions is usually represented as a maximum number of activated nodes or a number of iterations. After the algorithm terminates, activated nodes represent a similar nodes to the initial set of nodes.

Compared to our maximum activation method, the spreading activation does not guarantee an activation of a particular node while our method always assigns an activation if there exists an improving path between source and target nodes. Although there exist constrained spreading activation methods which utilise semantics of edges [6], no version of the spreading activation takes into account the "age" of edges as our method does. The maximum activation is better suited for the Web API selection mainly due to following reasons: 1) it is not known at which nodes the spreading activation terminates while the Web API selection problem uses Web API candidates as an input (target nodes), 2) the spreading activation has a local meaning of activations that indicates a measure that can be used for recommendations on data whereas maximum activation uses the value as a global measure of connectivity from source to target nodes.

There are other works in the area of Web Service discovery and selection including QoS selection [10,18], collaborative and content-based filtering methods [3,20,11,19] which are less relevant.

6 Conclusion and Future Work

A popularity and a growing number of Web APIs and mashups require new methods that users can use for more precise selection of Web APIs. Current approaches for searching and selecting Web APIs utilize rankings based on Web APIs popularity either explicitly expressed by users or a number of Web APIs used in mashups. Such metrics works well for the large, widely-known and well-established APIs such as Google APIs, however, they impede adoption of more recent, newly created APIs. In order to address this problem we proposed a novel activation-based Web API selection method which takes into account a user profile and user's preferences, temporal aspects (the creation time of Web APIs and mashups) and social links between users. While existing popularity-based rankings use a single-dimensional ranking criteria (i.e., a number of APIs used in mashups), our method uses multi-dimensional ranking criteria and with help of graph analysis methods it provides more precise results. The method requires a set of Web API candidates, a user profile and evaluates a ranking for all Web API candidates for the given user profile. The Web API candidates may result from a service discovery task that usually evaluates a match based on a coarse-grained search request. Service discovery requests may be represented as a functional category, for example, the discovery returns all services in the same category such as a mapping category.

In our future work we want to extend the method so that we can assign capacities to individual edges. In cooperation with ProgrammableWeb.com, we also plan to improve the Linked Web APIs dataset and eventually make it available in the Linked Data cloud. We want to enrich this dataset with user profiles from traditional social networks. We also plan to incorporate to our method various social network analysis metrics evaluated on the Linked Web APIs dataset. Last but not least we want to evaluate the method on datasets from the Linked Data cloud.

Acknowledgement. This work was supported by the Czech Technical University grant number SGS12/093/OHK3/1T/18 and the In-Coming Scholarship with application number 51100570 provided by the International Visegrad Fund. We also thank to ProgrammableWeb.com for supporting this research.

References

1. N. M. Akim, *et al.* Spreading activation for web scale reasoning: Promise and problems. In *WebSci*. 2011.
2. J. Al-Sharawneh and M.-A. Williams. A social network approach in semantic web services selection using follow the leader behavior. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, pp. 310 –319. sept. 2009.
3. F. Le andcue and. Combining collaborative filtering and semantic content-based approaches to recommend web services. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pp. 200 –205. sept. 2010.

4. J. R. Anderson. A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior*, 22:261–295, 1983.
5. S. Choudhury, J. Breslin, and A. Passant. Enrichment and ranking of the youtube tag space and integration with the linked data cloud. *The Semantic Web-ISWC 2009*, pp. 747–762, 2009.
6. F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11:453–482, 1997.
7. A. Dix, *et al.* Spreading activation over ontology-based resources: from personal context to web scale reasoning. *International Journal of Semantic Computing*, 4(1):59, 2010.
8. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399404, 1956.
9. A. Freitas, *et al.* Querying linked data using semantic relatedness: a vocabulary independent approach. *Natural Language Processing and Information Systems*, pp. 40–51, 2011.
10. M. Godse, U. Bellur, and R. Sonar. Automating qos based service selection. In *Web Services (ICWS), 2010 IEEE International Conference on*, pp. 534 –541. july 2010.
11. Y. Jiang, J. Liu, M. Tang, and X. Liu. An effective web service recommendation method based on personalized collaborative filtering. In *Web Services (ICWS), 2011 IEEE International Conference on*, pp. 211 –218. july 2011.
12. J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *Internet Computing, IEEE*, 11(6):60 –67, nov.-dec. 2007.
13. M. Shafiq, R. Alhajj, and J. Rokne. On the social aspects of personalized ranking for web services. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pp. 86 –93. sept. 2011.
14. R. Torres, B. Tapia, and H. Astudillo. Improving web api discovery by leveraging social information. In *Web Services (ICWS), 2011 IEEE International Conference on*, pp. 744 –745. july 2011.
15. T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *ESWC*, pp. 674–689. 2008.
16. T. Vitvar, S. Vinoski, and C. Pautaso. Programmatic interfaces for web applications, guest introduction (to appear). *IEEE Internet Computing*, Jul/Aug 2012.
17. S. Wang, X. Zhu, and H. Zhang. Web service selection in trustworthy collaboration network. In *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on*, pp. 153 –160. oct. 2011.
18. S. Yau and Y. Yin. Qos-based service ranking and selection for service-based systems. In *Services Computing (SCC), 2011 IEEE International Conference on*, pp. 56 –63. july 2011.
19. Q. Zhang, C. Ding, and C.-H. Chi. Collaborative filtering based service ranking using invocation histories. In *Web Services (ICWS), 2011 IEEE International Conference on*, pp. 195 –202. july 2011.
20. Z. Zheng, H. Ma, M. Lyu, and I. King. Wsrec: A collaborative filtering based web service recommender system. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pp. 437 –444. july 2009.