CS24w:
Social and Information Network Analysis

Assignment number: Final Project Report
Submission time: 11:59 PM PST and date: Dec 10, 2013

Fill in and include this cover sheet with each of your assignments. It is an honor code violation to write down the wrong time. Assignments are due at 9:30 am, either handed in at the beginning of class or left in the submission box on the 1st floor of the Gates building, near the east entrance.

Each student will have a total of two free late periods. One late period expires at the start of each class. (Homeworks are usually due on Thursdays, which means the first late periods expires on the following Tuesday at 9:30am.) Once these late periods are exhausted, any assignments turned in late will be penalized 50% per late period. However, no assignment will be accepted more than one late period after its due date.

Your name: Joe Fan
Email: joeFan@stanford.edu SUID: joeFan

Collaborators: Xavier Falco (xfalco) and Patrick Kelly (pkellyd)

I acknowledge and accept the Honor Code.

(Signed)

(For CS24w staff only)

Late periods: 1 2

<table>
<thead>
<tr>
<th>Section</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

Comments:
# HOMEWORK ROUTE FORM

**Stanford Center for Professional Development Student Information**

<table>
<thead>
<tr>
<th>Course No.</th>
<th>Faculty / Instructor Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS224W</td>
<td>Jure Leskovec</td>
<td>Dec 10, 2013</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe Fan, on behalf of Patrick Kelly and Xavier Falco</td>
<td>510-520-4307</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzy Logix</td>
<td><a href="mailto:joefan@stanford.edu">joefan@stanford.edu</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City.</th>
<th>State</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charlotte</td>
<td>NC</td>
<td>USA</td>
</tr>
</tbody>
</table>

Check One:  
- [ ] Homework #:  
- [ ] Midterm  
- [ ] Other  
  - Final project report

The email address provided on this form will be used to return homework, exams, and other documents and correspondence that require routing.

Total number of pages faxed including cover sheet: 41

### For Stanford Use Only

- **Date Received by the Stanford Center for Professional Development:**
- **Date Instructor returned graded project:**
- **Score/Grade:**  
  (to be completed by instructor or by teaching assistant)

Date the Stanford Center for Professional Development returned graded project:

---

**Please attach this route form to ALL MATERIALS and submit ALL to:**

Stanford Center for Professional Development  
496 Lomita Mall, Darand Building, Rm 410, Stanford, CA 94305-4036  
**Office** 650.725.3015 | **Fax** 650.725.4138  
For homework confirmation, email scpd-distribution@lists.stanford.edu  
http://scpd.stanford.edu

Last modified October 27, 2008
Adding Dimensions to Ranking in Social Graphs

Author: Xavier Falco
Author: Joe Fan
Author: Patrick Kellyl

December 10, 2013
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Literature</td>
<td>4</td>
</tr>
<tr>
<td>Community Structure</td>
<td>4</td>
</tr>
<tr>
<td>PageRank</td>
<td>4</td>
</tr>
<tr>
<td>ISPR</td>
<td>5</td>
</tr>
<tr>
<td>HITS</td>
<td>5</td>
</tr>
<tr>
<td>The Stack Overflow Network</td>
<td>6</td>
</tr>
<tr>
<td>Network features - questions, answers, users and tags</td>
<td>6</td>
</tr>
<tr>
<td>Network features - Community structure</td>
<td>7</td>
</tr>
<tr>
<td>Ranking Users with PageRank and HITS</td>
<td>10</td>
</tr>
<tr>
<td>Graphs: PageRank and HITS</td>
<td>10</td>
</tr>
<tr>
<td>Graphs: Intelligent Surfer PageRank</td>
<td>10</td>
</tr>
<tr>
<td>Success Metrics</td>
<td>11</td>
</tr>
<tr>
<td>Results</td>
<td>12</td>
</tr>
<tr>
<td>Conclusion</td>
<td>15</td>
</tr>
<tr>
<td>Appendix</td>
<td>16</td>
</tr>
<tr>
<td>Detailed Ranking Results: Graph 1</td>
<td>16</td>
</tr>
<tr>
<td>Detailed Ranking Results: Graph 2</td>
<td>16</td>
</tr>
<tr>
<td>PageRank code</td>
<td>17</td>
</tr>
<tr>
<td>HITS code</td>
<td>20</td>
</tr>
<tr>
<td>ISPR code</td>
<td>21</td>
</tr>
<tr>
<td>Testing code</td>
<td>25</td>
</tr>
<tr>
<td>Ingestion code</td>
<td>27</td>
</tr>
</tbody>
</table>
Introduction

In this paper, we explore information related to the behavior of users in social networks with network analysis concepts in order to uncover network structures, rank users, and glean insight into communities. Using data from StackOverflow (SO), the technical Question-Answer (Q&A) social network, we aim to answer questions such as:

- Who is the user best suited to answer a particular question?
- What is the specific area of expertise of user x?
- Are there implicit networks of users with different skill sets?

SO quantifies ranks users using a Reputation score, which is based on a number of parameters. Most of them, such as up votes received by users’ answers to questions, are based on skill; others are based on participation (e.g. editing a post, commenting) and site citizenship (reading tutorials, providing incentives to others). The site provides a breakdown of users reputation by topic in a user profile page.

Our work touches on three areas. Firstly, we developed some basic insight into the graph structure to familiarize ourselves with the SO network. Secondly, we outlined user and topic clusters at an aggregate level. Finally, we sought to score users’ skills with both a targeted and Intelligent Surfer version of PageRank and a variant of the HITS algorithm.

We conclude that:

**Power law distributions pervade the SO network**
Indeed, they adequately describe respondents per question, reputation score frequency, frequency of number of tags per question, and PageRank influence rank.

**Topical and user clusters are identifiable**
Defining proximity as being proportional to the number of shared links, the resulting 2D layout places platform-specific (i.e. .Net, IPhone) users/concepts in distant locations while identifying more generic topics such as programming languages closer to the center. Topics common to most development challenges such as images and parsing are also found mostly in the center.

**PageRank and HITS provide a skill score that is more accurate than SO Reputation**
Both topic-agnostic and topic-specific PageRank approaches, and HITS, are better at identifying the users that will provide the best answers given a specific question.
A number of exciting areas for further work, ideally suited to the subject matter and graph structure of the SO site, were identified on a preliminary level:

- Firstly, we would like to explore overlaps in various tag-based communities in Stack Overflow on a more granular basis. It is interesting to note that the community we studied presented a small group of power users with high influence score for each algorithm tested. An intriguing avenue of exploration would explore whether those same users may be power users in other communities. Are the ”Stack Overflow” experts ( i.e. those users with high influence ranking on the network consisting of all of Stack Overflow ) users who are also experts across a variety of communities, or are those users distinct from the expert group of smaller communities?

- Secondly, we would like to use question text and code snippets in combination with PageRank to obtain more specific skill scores. Indeed, our results show that a page rank model, while more effective than the SO reputation score, is in some respect an incomplete metric of user influence. With a more granular understanding of the quantitative data in the network, such as the textual metadata found from questions and answers, we can hope to further our understanding of the making of the influence flow in a Stack Overflow community.

- Finally, we would like to develop a metric to measure topical distance between nodes and/or users. This could be used to identify the path a user should follow to learn new technologies in a coherent fashion, helping users acquire new concepts and technologies. Alternatively, we could with this approach identify user groups (single skill, multi-skill) that can be targeted for professional or commercial purposes.
Literature

Community Structure

Several approaches to the identification of communities have been proposed. A community can be defined as a set of highly interconnected nodes that is sparsely linked to the rest of the network. To detect communities such as these Girvan and Newman use edge betweenness in combination with modularity to segment a graph \(^1\). This paradigm is suited to situations where geography is a relevant influence on graph structure, for instance, a sports league comprised of numerous teams, each based at given location. Intra-regional links will be pervasive and strong (local teams constantly compete amongst each other) and inter-regional links will be sparser and weaker (perhaps only local champions are able to contend in competitions where teams from various regions participate). In contrast, Palla, et al. \(^2\) models highly overlapping communities by defining the probability of belonging a given set of communities as dependent on the product of the probabilities of belonging to each separately. In this case we can imagine a graph with members of only one community on the fringe and members of higher degrees of ”cosmopolitanism” (defined as the number of communities a node belongs to) approaching the center, where multiple communities overlap as ”tiles”. Later we will show that the SO network shows both of these types of community structure, depending on if you take a topical or user based view.

PageRank

PageRank is an algorithm that assigns to each node in a directed graph a numerical weighting. It was popularized as the backbone of the popular search engine Google \(^3\), and attempts to derive for each node in a network the expected percentage of time that of a web surfer randomly following links from page to page will spend on that particular node. The algorithm may be applied to any collection of nodes linked by directed edges. Rafiei and Mendelzon \(^4\) create a variant PageRank by adding a topical dimension. Their approach computes a given page’s reputation on a specific topic, and, conversely, can return on which specific topics a given page has the highest reputation. They apply a similar extension to the HITS algorithm (explained below). The work of Rafiei and Mendelzon permits new treatment of directed graph models that attach metadata to either nodes or edges. By considering entries in the metadata of a node or in the union of the metadata of its outgoing edges, we can assign to each node a set of values along a new dimension that parallels the setup of web pages and topics in the study discussed. This extension of the graph can permit a richer understanding of its structure.

\(^1\)M. Girvan, M.E.J. Newman, Community structure in social and biological networks
\(^4\)R. Rafiei, A. Mendelzon, What is this page known for ? Computing Web Page Reputations
Domingos and Richardson\textsuperscript{5} provide an alternative approach to add the topic dimension to PageRank. Instead of using a purely random walk approach as PageRank does, their \textit{Intelligent Surfer} progresses to a node with probability dependent on the relevance of that node to the topics of the search query. To implement this, numerous PageRank vectors are computed and combined at query time as per the composition of the query.

The Intelligent Surfer PageRank (ISPR) algorithm modifies the PageRank algorithm to take into account the content of each node, which is represented in each Stack Overflow question by tags. ISPR was proposed to improve Internet webpage search, and we implemented the algorithm and tuned it to fit the Stack Overflow graph. In vanilla PageRank, we calculate a vector that contains the PageRank for every node. The basic idea of ISPR is to calculate a PageRank vector for each query term \( q \) in the corpus. For each query term \( q \), the probability of going from the current node \( i \) to a neighbor node \( j \) is dependent on the relevance \( R \) of node \( j \) to the query term \( q \) (denoted \( R_q(i \rightarrow j) \)). As in vanilla PageRank, there is a small chance that the user jumps to a random node, and in ISPR these nodes must contain the query term \( q \). So now we have calculated a PageRank vector for every query term in the corpus in batch. When an actual search query is submitted to the system, ISPR calculates a custom PageRank vector by using the words in the search query, which is a linear summation of the cached PageRank vector of each query word.

We test both the plain vanilla PageRank and the Intelligent Surfer models as a part of this project.

**HITS**

The HITS algorithm, developed by Kleinberg, et al.\textsuperscript{6}, assigns a \textit{hub} and an \textit{authority} score to each page for each topic. The former is proportional to the sum of the authority score on that topic of outbound pages. The latter is proportional to the sum of the hub score on that topic of inbound pages. The reputation of a page on a topic is proportional to its authority and hub score on that topic. The algorithm was designed for an early model of the web in which certain pages gravitated towards a "hub" model, producing value by linking to and aggregating directories of high worth pages while others gained reputation from their ranks in such "hub" pages by providing valuable content. Over time, as the model of the web has shifted, this algorithm has not been as popular as the PageRank model. Nonetheless, its particular fit for graphs of a dichotomous nature makes it a particularly interesting candidate for social networks such as Q&A sites, in which entities in the network (such as users) are generally heavily sided towards either the \textit{question} side or the \textit{answer} side. In this research, we explore whether an adaption of the HITS algorithm can help determine expertise in various subnetworks of the Stack Overflow dataset.

\textsuperscript{5}P. Domingos, M. Richardson. \textit{The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank}. In Proc. NIPS, 2002

\textsuperscript{6}J. Kleinberg. \textit{Authoritative sources in a hyperlinked environment}. Proc. 9th ACM-SIAM Symposium on Discrete Algorithms, 1998
The Stack Overflow Network

*Stack Overflow* users post questions on the SO site and other users answer these questions. Both questions and answers can receive *Up Votes* which are issued by users that believe the question is of high quality and or *Down Votes* by those who think the contrary. Answers are listed in an order defined by the Net votes each answer has, defined as the up-votes minus the down-votes. Questions have a title, a body (which usually contain contains code snippets and output), and *tags* - labels that characterize or summarize the nature of the question. A user’s reputation depends largely on the Net votes received by answers and questions posted by him.

Network features - questions, answers, users and tags

To gain an understanding of the inner workings of the SO network, we developed basic metrics relating to users, questions, answers, and reputation. As one would expect, all these features follow a power law as depicted in the following figures.
Indeed, the SO network displays a structure typical of a social network, and could be simulated though a preferential attachment-type model.

**Network features - Community structure**

We looked at community structure from two standpoints: that of topics and that of users. In analyzing topics, we first developed an understanding of the probability that a topic has coexisting with others in a question. The following figure depicts the correlation of the top topics. We can clearly see, for instance, that platform-related topics tend to appear together. Indeed, the bright colors in the off diagonal cells are be in the same cluster as the diagonal element in the same row or column.

The figure that follows is a 2D representation of the graph, plotting topics as nodes and placing edges between topics if they coexist in some question. We used a Gephi Force Map layout which disposes more inter-linked nodes closer to each other and those less so farther apart. In this representation it is evident that, for instance, platform and technology-specific topics are on the fringe, and as we progress toward the center we see more generic topics, culminating in completely technology agnostic items (e.g., ”image”, ”performance”). We highlight some important topic groups. Note the overlap between groups also has meaning: for instance, web technologies intersect both Microsoft and Open technologies and cover the expected topics.
Indeed, this topical view of the graph may be interpreted as displaying a structure similar to the highly overlapping communities described by Palla, et al. This is sensible, as developers, regardless of their area, will deal with a wide range of topics, from the very platform specific to the more generic. For instance, only a java developer will deal with java syntax, but all developers will deal with many common topics like "database", "XML", and "debugging".
The figure above shows users as nodes. Each user is labeled per tags related to questions they have answered. Users are linked to all other users with whom they share tags. We highlight some of the main user clusters. On the fringes we see those related to mainstream, roughly mutually exclusive technologies or platforms. Users in the center area will be concerned more with general topics in programming such as theory, algorithms, and other conceptual topics. So you may expect the expert programmer to be in a peripheral community and an academic theorist to be in the middle of the graph. Viewed in a user-centric way, the SO network appears more like a set of dense communities that are sparsely interlinked. This may mean that users that develop for a given platform or that use a specific language deal mainly with each other on common issues and challenges, and users that see questions tagged with languages or platforms different from the ones they use, may not consider answering them even if there is a possibility that they have a valid contribution.
Ranking Users with PageRank and HITS

We now turn our attention to alternative methods of ranking user skills. We apply PageRank, HITS and the Intelligent Surfer approach on two variations of the SO graph.

Graphs: PageRank and HITS

From the stack overflow database, we chose to focus our attention to a targeted sub-network consisting of all questions containing a reference to the ”python” tag to measure the effectiveness of various algorithms in ranking user influence and expertise. We generated this subgraph using two approaches.

In the first approach, we considered as nodes only users from ”python” network. Each unit of net positive upvote for an answer to a question resulted in a directed edge from the user corresponding to the question to the user corresponding to the answer. Note that this often resulted in multiple weighted edges between two nodes. This allowed weighing a particular edge in the algorithms we tested, which gave variants of the algorithms studied that were better suited to the nature of the graphs we were studying (networks for which we have quantitative information on not just the association between two nodes, but also on the strength of the association). This resulted in 471,285 distinct source-destination user combinations. The data contained reference to 114,833 users (73,484 question-posters and 65,434 answer-posters) and 446,438 responses. Thus our first graph had 114,833 nodes and 471,285 edges (obviating redundancies which are later consolidated).

In the second approach, we considered a more bipartite structure with a stronger delineation between question and answer. Under this schema, we mapped each question id and each user id to a separate node. Edges were generated from question ids to user ids, and were also created for each unit of net positive upvote for an answer to a question. This created a bipartite graph structure, which was a stronger representation of the qualitative structure of the Stack Overflow network, given its Q&A nature.

Graphs: Intelligent Surfer PageRank

For the Stack Overflow graph, the implementation of ISPR is consistent with the Richardson paper and is also optimized to address the structure of the Stack Overflow graph. The Stack Overflow tags correspond to the query terms in Internet webpage search. There are two key associations between ISPR and Stack Overflow. First, the Stack Overflow question’s tags are appended to the edges between the questioner and each answerer. Since Stack Overflow questions can contain only 0 or 1 instance of each tag, we realize that $R_q(i \rightarrow j)$ is binary. This observation allows us to create subgraphs for each tag that contain only nodes related to that tag. We can then run PageRank for each of these subgraphs, which dramatically improves the computational speed while retaining accuracy. This approach can be scaled to run in parallel in a large compute cluster.

The second association is that each Stack Overflow question contains several tags, which correspond to the query words in a search query in Internet search. For each Stack Overflow question, we dynamically calculate the linear summation of PageRank vector of each tag. The weight of each tag affects the ranking accuracy. A naive weighing is to simply average the PageRank vectors. We dramatically improved the performance by using the inverse document frequency as the weight. Our results going forward will reflect these associations.
Success Metrics

We compared each algorithm’s ability to predict the order of users’ answers. The comparison was done using two distinct metrics, one measuring the predictability of the top answer and the other the predictability of answer orders. We also varied the scope of the comparison by restricting it to questions with a minimum number of net up votes. This makes sense as there may be a lot of noise in questions which have received little attention from the community and it is the collective attention of the community that brings discrimination amongst users by skill level.

Specifically, the comparison analysis was performed along the following metrics and scopes:

**Metric 1, top answer** Defined as the percentage of times that the "best" answer to a question (i.e. the one with the most net up votes) was authored by the user who had the highest ranking across all users who answered.

**Metric 2, all answers** Defined as the percentage of times that the ranking of two answers for a given question matched the placement of their authors on the ranking. The metric tallied the following:

- An answer A1 by user U1 to some question Q and an answer A2 by user U2 to Q were such that A1 had more net up votes than A2 but U2 had higher ranking than U1 (this would be a "wrong" tally by the ranking being tested).
- An answer A1 by user U1 to some question Q and an answer A2 by user U2 to Q were such that A1 had more net up votes than A2 and U1 had higher ranking than U2 (this would be a "correct" tally by the ranking being tested).

**Cut off** for a ranking was defined as the minimum net up vote needed by an answer to qualify as a data point for both of the above metrics.
Results

The following tables summarize results delivered by each approach. We include the results for simulations using cutoffs values of 1 through 30 net up votes, as ranges beyond this provided very few data samples. A reasonable upper bound of the cutoff is around 30 because very few questions have more than 30 answers. We observed that HITS and ISPR outperformed the other methods in this range. In comparing the pair wise relevance ranking for only the top answerer, ISPR was quite strong. The tag specific page ranks of each user was instrumental in customizing the ranking to identify the top user. The performance of the other non-tag specific methods were lower and similar to each other. When comparing the ranking across all users, the value of ISPR shines since the performance is even better than just identifying the top user. This shows that ISPR works well across all rankings. Detailed results are included in the appendix

<table>
<thead>
<tr>
<th>Metric 1: Top Answer</th>
<th>SO</th>
<th>PR</th>
<th>HITS</th>
<th>ISPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph 1: People &gt; People</td>
<td>49.86</td>
<td>50.49</td>
<td>50.08</td>
<td>59.16</td>
</tr>
<tr>
<td>Graph 2: Questions &gt; People</td>
<td>49.86</td>
<td>50.49</td>
<td>49.05</td>
<td>not run</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric 2: All Answers</th>
<th>SO</th>
<th>PR</th>
<th>HITS</th>
<th>ISPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph 1: People &gt; People</td>
<td>57.63</td>
<td>57.00</td>
<td>71.33</td>
<td>64.55</td>
</tr>
<tr>
<td>Graph 2: Questions &gt; People</td>
<td>57.63</td>
<td>57.00</td>
<td>71.24</td>
<td>not run</td>
</tr>
</tbody>
</table>

- For the prediction of the top answer the ISPR is the superior approach by a large margin. On both graph structures. On graph 1, both PageRank and HITS outperform SO Reputation. This is explainable by the fact that the page rank and hits in fact correspond to the principal eigenvector of the adjacency matrix. Since ISPR optimizes the weights of the tag-specific page rank, it performs better for questions with many tags. In fact, for questions that have ≥1 tag, ISPR doubly optimizes the ranking at the top

- For the prediction of overall answer order, HITS is superior on both graphs. ISPR (run only on graph 1) performed well, and PageRank fared worse than SO Reputation HITS internally models a dichotomic flow of influence, which is evidently an effective modeling tool for the dual question and answer nature of the SO network.

Both PageRank and HITS distributions obey power laws:
In Graph 1, PageRank seems to define an upper bound to Reputation in the Python subgraph. Thus we find users with high SO Reputation and low PageRank. This corresponds to users that either have developed their reputation on other topics, far removed from python, or users that have developed a large proportion of their reputation presenting questions (outlinks) rather than answers (inlinks).

The relationship between HITS and PageRank and between HITS and SO Reputation is slightly more cryptic. High HITS scores seem to correlate in a positive way with both of the other metrics. For lower hits scores this is reversed. Further study is required to understand this behavior.
Now we look at the behavior of these metrics in terms of how many questions and answers different users posted. The value of the ranking of each user is presented as the color of the corresponding data point. (Note that the top score is Rank 0: black) We see here that PageRank, given a same number of answers, will assign less score to frequent askers of questions, as each question asked is an outflow of rank. The opposite is true for HITS and SO Reputation. For the same number of questions answered, asking mode seems to improve score. In the case of SO reputation this is understandable as questions also contribute to Reputation. In the case of HITS, this means that asking questions raises the hub component of your overall reputation.

More questions hurt score  Questions and answers contribute to score.
Conclusion

Our stated objective was to explore dimensions in social graphs beyond those implied by explicit links and rankings using network analysis tools and concepts. We managed to do this on the SO graph by developing single and multi topic skill scores to users with variants of the PageRank, Intelligent Surfer and HITS methodologies. We also defined topical and user communities based on question content. As we worked on the Stack Overflow network, we can state learnings specific to it:

- The SO network displays a typical social graph structure, where power laws govern most variables. This includes items such as questions per user, answers per question, reputation distribution. SO could probably be modeled using the preferential attachment model.

- A topical graph with nodes representing topics and links representing their coexistence in some question, can be viewed as an overlapping communities graph, if communities are defined by areas of user activity such as website development or Microsoft Windows application development.

- A users graph with nodes representing users and links representing the fact they have answered questions with like tags, can be considered a group of densely connected communities with sparse links among them.

- Graph node scoring tools are a valid way of developing scores representing user skills. Specifically PageRank, HITS and their topic specific variations can be used to predict user performance in question more precisely than generic SO reputation can.

- Specifically, ISPR is an excellent tool to predict the user best suited to respond a given question. HITS is a better predictor of overall order as it discriminates better between less skilled users.
## Appendix

### Detailed Ranking Results: Graph 1

<table>
<thead>
<tr>
<th>cut off</th>
<th>SO</th>
<th>PR</th>
<th>HITS</th>
<th>ISPR</th>
<th>SO</th>
<th>PR</th>
<th>HITS</th>
<th>ISPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50.32</td>
<td>50.83</td>
<td>49.52</td>
<td>56.69</td>
<td>58.83</td>
<td>59.61</td>
<td>61.49</td>
<td>63.87</td>
</tr>
<tr>
<td>6</td>
<td>49.18</td>
<td>50.48</td>
<td>50.08</td>
<td>56.39</td>
<td>57.26</td>
<td>57.96</td>
<td>67.91</td>
<td>62.38</td>
</tr>
<tr>
<td>11</td>
<td>49.87</td>
<td>49.65</td>
<td>50.72</td>
<td>59.73</td>
<td>57.42</td>
<td>56.73</td>
<td>71.34</td>
<td>63.65</td>
</tr>
<tr>
<td>16</td>
<td>50.34</td>
<td>50.41</td>
<td>50.27</td>
<td>62.42</td>
<td>58.80</td>
<td>57.38</td>
<td>73.53</td>
<td>67.49</td>
</tr>
<tr>
<td>21</td>
<td>50.47</td>
<td>50.47</td>
<td>49.12</td>
<td>64.58</td>
<td>58.82</td>
<td>56.73</td>
<td>73.94</td>
<td>70.12</td>
</tr>
<tr>
<td>26</td>
<td>48.49</td>
<td>49.50</td>
<td>48.92</td>
<td>57.14</td>
<td>57.16</td>
<td>55.42</td>
<td>74.75</td>
<td>65.31</td>
</tr>
<tr>
<td>31</td>
<td>50.38</td>
<td>52.12</td>
<td>51.92</td>
<td>57.14</td>
<td>55.11</td>
<td>55.20</td>
<td>76.37</td>
<td>59.02</td>
</tr>
<tr>
<td>36</td>
<td>50.86</td>
<td>53.55</td>
<td>54.28</td>
<td>58.82</td>
<td>55.20</td>
<td>55.50</td>
<td>78.60</td>
<td>57.14</td>
</tr>
<tr>
<td>41</td>
<td>50.77</td>
<td>52.00</td>
<td>54.46</td>
<td>60.00</td>
<td>54.03</td>
<td>53.90</td>
<td>80.07</td>
<td>57.89</td>
</tr>
<tr>
<td>46</td>
<td>50.57</td>
<td>52.83</td>
<td>54.34</td>
<td>54.55</td>
<td>54.44</td>
<td>53.60</td>
<td>81.16</td>
<td>53.33</td>
</tr>
<tr>
<td>51</td>
<td>52.47</td>
<td>53.81</td>
<td>56.05</td>
<td>52.63</td>
<td>55.29</td>
<td>55.58</td>
<td>81.47</td>
<td>51.85</td>
</tr>
<tr>
<td>56</td>
<td>52.31</td>
<td>52.82</td>
<td>55.38</td>
<td>53.33</td>
<td>53.38</td>
<td>52.85</td>
<td>81.92</td>
<td>57.89</td>
</tr>
<tr>
<td>61</td>
<td>51.55</td>
<td>52.17</td>
<td>54.66</td>
<td>70.00</td>
<td>53.82</td>
<td>52.96</td>
<td>82.51</td>
<td>70.00</td>
</tr>
<tr>
<td>66</td>
<td>50.36</td>
<td>51.09</td>
<td>55.47</td>
<td>71.43</td>
<td>54.07</td>
<td>53.66</td>
<td>82.93</td>
<td>71.43</td>
</tr>
<tr>
<td>71</td>
<td>51.22</td>
<td>52.03</td>
<td>53.66</td>
<td>80.00</td>
<td>54.57</td>
<td>52.71</td>
<td>82.79</td>
<td>80.00</td>
</tr>
<tr>
<td>76</td>
<td>49.55</td>
<td>51.35</td>
<td>50.45</td>
<td>80.00</td>
<td>54.35</td>
<td>51.67</td>
<td>79.60</td>
<td>80.00</td>
</tr>
<tr>
<td>81</td>
<td>50.49</td>
<td>52.43</td>
<td>50.49</td>
<td>80.00</td>
<td>55.00</td>
<td>52.24</td>
<td>80.69</td>
<td>80.00</td>
</tr>
<tr>
<td>86</td>
<td>52.75</td>
<td>53.85</td>
<td>51.65</td>
<td>75.00</td>
<td>54.53</td>
<td>50.98</td>
<td>80.71</td>
<td>75.00</td>
</tr>
<tr>
<td>91</td>
<td>51.81</td>
<td>54.22</td>
<td>53.01</td>
<td>66.67</td>
<td>54.67</td>
<td>51.02</td>
<td>81.30</td>
<td>66.67</td>
</tr>
</tbody>
</table>

### Detailed Ranking Results: Graph 2

<table>
<thead>
<tr>
<th>cut off</th>
<th>SO</th>
<th>PR</th>
<th>HITS</th>
<th>ISPR</th>
<th>SO</th>
<th>PR</th>
<th>HITS</th>
<th>ISPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50.32</td>
<td>50.83</td>
<td>48.18</td>
<td>56.69</td>
<td>58.83</td>
<td>59.61</td>
<td>61.41</td>
<td>63.87</td>
</tr>
<tr>
<td>6</td>
<td>49.18</td>
<td>50.48</td>
<td>49.02</td>
<td>56.39</td>
<td>57.26</td>
<td>57.96</td>
<td>68.16</td>
<td>62.38</td>
</tr>
<tr>
<td>11</td>
<td>49.87</td>
<td>49.65</td>
<td>49.80</td>
<td>59.73</td>
<td>57.42</td>
<td>56.73</td>
<td>71.61</td>
<td>63.65</td>
</tr>
<tr>
<td>16</td>
<td>50.34</td>
<td>50.41</td>
<td>48.58</td>
<td>62.42</td>
<td>58.80</td>
<td>57.38</td>
<td>73.16</td>
<td>67.49</td>
</tr>
<tr>
<td>21</td>
<td>50.47</td>
<td>50.47</td>
<td>48.70</td>
<td>64.58</td>
<td>58.82</td>
<td>56.73</td>
<td>73.40</td>
<td>70.12</td>
</tr>
<tr>
<td>26</td>
<td>48.49</td>
<td>49.50</td>
<td>48.49</td>
<td>57.14</td>
<td>57.16</td>
<td>55.42</td>
<td>74.75</td>
<td>65.31</td>
</tr>
<tr>
<td>31</td>
<td>50.38</td>
<td>52.12</td>
<td>50.58</td>
<td>57.14</td>
<td>55.11</td>
<td>55.20</td>
<td>76.37</td>
<td>59.02</td>
</tr>
<tr>
<td>36</td>
<td>50.86</td>
<td>53.55</td>
<td>55.55</td>
<td>58.82</td>
<td>55.20</td>
<td>55.50</td>
<td>78.60</td>
<td>57.14</td>
</tr>
<tr>
<td>41</td>
<td>50.77</td>
<td>52.00</td>
<td>53.85</td>
<td>60.00</td>
<td>54.03</td>
<td>53.90</td>
<td>78.98</td>
<td>51.85</td>
</tr>
<tr>
<td>46</td>
<td>50.57</td>
<td>52.83</td>
<td>55.09</td>
<td>54.55</td>
<td>54.44</td>
<td>53.60</td>
<td>80.32</td>
<td>57.89</td>
</tr>
<tr>
<td>51</td>
<td>52.47</td>
<td>53.81</td>
<td>55.61</td>
<td>52.63</td>
<td>55.29</td>
<td>55.58</td>
<td>80.06</td>
<td>70.00</td>
</tr>
<tr>
<td>56</td>
<td>52.31</td>
<td>52.82</td>
<td>54.87</td>
<td>53.33</td>
<td>53.38</td>
<td>52.85</td>
<td>80.23</td>
<td>66.67</td>
</tr>
<tr>
<td>61</td>
<td>51.55</td>
<td>52.17</td>
<td>54.66</td>
<td>70.00</td>
<td>53.82</td>
<td>52.96</td>
<td>80.54</td>
<td>57.89</td>
</tr>
<tr>
<td>66</td>
<td>50.36</td>
<td>51.09</td>
<td>55.47</td>
<td>71.43</td>
<td>54.07</td>
<td>53.66</td>
<td>80.76</td>
<td>71.43</td>
</tr>
<tr>
<td>71</td>
<td>51.22</td>
<td>52.03</td>
<td>55.28</td>
<td>80.00</td>
<td>54.57</td>
<td>52.71</td>
<td>80.76</td>
<td>80.00</td>
</tr>
<tr>
<td>76</td>
<td>49.55</td>
<td>51.35</td>
<td>54.95</td>
<td>80.00</td>
<td>54.35</td>
<td>51.67</td>
<td>78.09</td>
<td>70.12</td>
</tr>
<tr>
<td>81</td>
<td>50.49</td>
<td>52.43</td>
<td>55.34</td>
<td>80.00</td>
<td>55.00</td>
<td>52.24</td>
<td>78.79</td>
<td>59.02</td>
</tr>
<tr>
<td>86</td>
<td>52.75</td>
<td>53.85</td>
<td>57.14</td>
<td>75.00</td>
<td>54.53</td>
<td>50.98</td>
<td>78.54</td>
<td>80.00</td>
</tr>
<tr>
<td>91</td>
<td>51.81</td>
<td>54.22</td>
<td>56.63</td>
<td>66.67</td>
<td>54.67</td>
<td>51.02</td>
<td>78.66</td>
<td>66.67</td>
</tr>
</tbody>
</table>
PageRank

#!/usr/bin/env python

# # # # # # # # # # # # # # # # # # # # # # # # # # #
# PAGERANK IMPLEMENTATION
# v2 – With weighted edges
# # # # # # # # # # # # # # # # # # # # # # # # # # #
# In this version:
# – The float attribute of nodes is their pageRank – hence the PageRank hashtable
# has been eliminated
# – The edge weights reside in the float attribute of edges

import snap
import time

def simple_graph():
    # Generates a simple graph for testing (cs246 book, chapter 5...)
    # Graph in this version has weighted edges...
    # Note that the weight attribute can only be initialized once the
    # edges are created
    g = snap.TNEANet.New()
    for i in range(4):
        g.AddNode(i+1)
    g.AddEdge(1, 2)
    g.AddEdge(1, 3)
    g.AddEdge(1, 4)
    g.AddEdge(2, 1)
    g.AddEdge(2, 3)
    g.AddEdge(3, 2)
    g.AddEdge(3, 4)
    g.AddEdge(4, 1)
    g.AddFltAttrE("EValFlt", 1.0)
    g.AddFltAttrN("NValFlt", 0.0)
    # AddWEdge(g, 1, 2, 50)
    # AddWEdge(g, 1, 3, 1)
    # AddWEdge(g, 1, 4, 1)
    # AddWEdge(g, 2, 3, 2)
    # AddWEdge(g, 3, 4, 3)
    return g

def AddWEdge(graph, src_node_id, dst_node_id, weight = 1.0):
    # adds weight to an edge
    edge_id = graph.GetEId(src_node_id, dst_node_id)
    graph.AddFltAttrDatE(edge_id, weight, "EValFlt")
    return

def total_difference(HashTable1, HashTable2):
    # Returns 'difference' between the values of two hash tables
    # It's calculated as the sum of absolute values of the difference between corresponding items
    # Use it to compare the outputs of the SNAP PageRank and our python version
    difference = 0.00
    for i in HashTable1:
        difference += abs(i.GetDat() - HashTable2.GetDat(i.GetKey()))
    return difference

def initialize_PR_vector_w(graph):
    # Sets the initial PR value of all nodes to 1/|N|
    initial_PR_score = 1.0/graph.GetNodes()
    for n in graph.Nodes():
        graph.AddFltAttrN(n.GetId(), 1, "NValFlt")
    return

def pre_process(graph):
    # Build a dictionary with proportion of PR that goes out of every edge of every node
    # Example: if a node has 2 out links, one with weight 1 and the other with weight 2
    # then 1/3 of the PR will be passed on through the first link and 2/3 through the other
    edge_PR_ratio = {}  
    for e in graph.Edges():
        src = e.GetSrcNId()
        dst = e.GetDstNId()
        edge_PR_ratio[e.GetId()] = get_w_out_PR_ratio(graph, src, dst)
return edge_PR_ratio
def test_in_node_PR_weight():
    graph = simple_graph()
    PRankH = initialize_PR_vector(graph)
correct_results = [0.375, 0.20833333, 0.20833333, 0.20833333]
    error = 0
    for r in range(4):
        print get_in_node_PR_weight(graph, r + 1, PRankH, 1), correct_results[r]
        error = error + get_in_node_PR_weight(graph, r + 1, PRankH) - correct_results[r]
    if error < 0.00001:
        print "test ok - error < 0.00001 (%)" % (error)
    else:
        print "test not ok - error > 0.00001 (%)" % (error)
def get_in_node_PR_weight(graph, edge_PR_ratio, node_id):
    # Returns the PR weight that 'reaches' a given node based on inlinks
    # No 'leakage' is considered here
    Node = graph.GetNI(node_id)
in_degree = Node.GetInDeg()
    pr_weight = 0.0
    for r in range(in_degree):
        in_node_id = Node.GetInNId(r)
        in_node_pr = graph.GetFltAttrDatN(in_node_id, "NValFlt")
        in_link_pr_contribution = edge_PR_ratio * in_node_pr
        pr_weight = pr_weight + in_link_pr_contribution
    return pr_weight
def get_out_PR_ratio(graph, src_node_id, dst_node_id):
    # Returns the weighted PageRank to be transferred from the specified source node to the specified destination node
    source_node = graph.GetNI(src_node_id)
total_weight = 0.0
    pr_weight = 0.0
    # Step 1: Calculate the sum of the weight of all outgoing edges
    for out_edge in source_node.GetOutEdges():
        edge_id = graph.GetEId(src_node_id, out_edge)
total_weight += graph.GetFltAttrDatE(edge_id, "EValFlt")
    # Step 2: Get the weight of the edge linking the source to the destination nodes
    edge_id = graph.GetEId(src_node_id, dst_node_id)
edge_weight = graph.GetFltAttrDatE(edge_id, "EValFlt")
    # Step 3: calculate the PR considering weights
    try:
        pr_weight = edge_weight / total_weight
    except ZeroDivisionError:
        print "error --> div by zero in edge: src_node_id, dst_node_id"
    return pr_weight
def RW_iteration_w(graph, edge_PR_ratio, C = 0.85):
    # Performs one Random Walk
    PRankH_temp = snap.TIntFltH()
    # Step 1: calculate new page ranks from in-nodes
    # This new page rank is 'dampened' by a factor C, usually about 0.85
    for n in graph.Nodes():
        node_id = n.GetId()
        PR = get_in_node_PR_weight(graph, edge_PR_ratio, node_id)
        PRankH_temp.AddDat(node_id, PR * C)
    # Step 2: The total rank lost to leakage is calculated (sum)
    # The leaked value is then apportioned to all nodes by adding leakage/|N| to each node
    sum = diff = NewVal = 0.0
    for i in PRankH_temp:
        sum += i.GetDat()
    leaked = (1 - sum) / float(graph.GetNodes())
    for n in graph.Nodes():
        NewVal = PRankH_temp(n.GetId()) + leaked
        OldVal = graph.GetFltAttrDatN(n.GetId(), "NValFlt")
        diff += abs(OldVal - NewVal)
        graph.AddFltAttrDatN(n.GetId(), NewVal, "NValFlt")
    print diff
# Return value is the 'difference' in value between the new PRs and the old ones.
# After this value goes below some threshold, we will want to stop iterating random walks

return diff

def GetPageRankPy_w(graph, C=0.85, Eps = 1e-4, MaxIter = 100):
    # This is (our) the Python page rank function
    # Calling is analogous to the SNAP function — you pass the graph and a hash table
    # and the function changes the hash table to reflect the PageRank
    # C is the damping factor
    # When difference between new ranks and old ones is less than Eps, we stop walking
    # Note that if you adjust Eps, C or MaxIter, your results will differ from SNAP's
    # Step 0: Add attributes to nodes (to hold PageRank) and edges
    # (to hold edge weights)
    # Also, build a dictionary with proportion of PR that goes out of every edge of every node
    edge_PR_ratio = pre_process(graph)
    # Step 1: all nodes have a page rank of 1/|N|
    initialize_PR_vector_w(graph)
    # Step 2: We perform random walks till page ranks don't change or we iterate MaxIter times
    for r in range(MaxIter):
        print "iteration ", r
        difference = RW_iteration_w(graph, edge_PR_ratio, C)
        if difference < Eps:
            break
    return
HITS

#! /usr/bin/env python
# HITS IMPLEMENTATION
# @author x falco

import snap
import time
from math import fabs, sqrt

def init(graph, users, hubs, authorities, verbose):
    NUMQUESTIONS = len(graph)
    initHub = 1.0 / sqrt(NUMQUESTIONS)
    for questionId in graph:
        hubs[questionId] = initHub
    for userId in graph[questionId]:
        users[userId] = {}
        users[userId][questionId] = graph[questionId][userId]
    NUMUSERS = len(users)
    initAuthority = 1.0 / sqrt(NUMUSERS)
    for userId in users:
        authorities[userId] = initAuthority
    if verbose:
        print "Done initializing : %d users, %d questions" % (NUMUSERS, NUMQUESTIONS)

def diff(authorities, oldAuthorities, verbose):
    diffW = 0.0
    numOldAuthorities = len(oldAuthorities)
    for userId in authorities:
        if not numOldAuthorities == 0:
            diffW += fabs(authorities[userId] - oldAuthorities[userId])
        else:
            diffW += authorities[userId]
    oldAuthorities[userId] = authorities[userId]
    return diffW

def doIteration(authorities, hubs, graph, users, verbose):
    newAuthorities = {}
    newHubs = {}
    # do new authorities
    for userId in users:
        weight = 0.0
        for questionId in users[userId]:
            weight += users[userId][questionId] * hubs[questionId]
        newAuthorities[userId] = weight
    # do new hubs
    for questionId in graph:
        weight = 0.0
        for userId in graph[questionId]:
            weight += graph[questionId][userId] * authorities[userId]
        # normalizes
        sumAuthorities = sum(newAuthorities.values())
        sumHubs = sum(newHubs.values())
        if verbose:
            print "Post-normalization sums: %s authorities, %s hubs" % (sumAuthorities, sumHubs)
        for questionId in newHubs:
            hubs[questionId] = newHubs[questionId] / sumHubs
        for userId in newAuthorities:
            authorities[userId] = newAuthorities[userId] / sumAuthorities
    if verbose:
        print "(Iteration %d) : %f diff" % (numIteration, diffW)
    return authorities

def calculateRankings(graph, verbose=True, Eps=1e-4):
    users = {}
    authorities = {}
    oldAuthorities = {}
    init(graph, users, hubs, authorities, verbose)
    numIteration = 0
    diffW = diff(authorities, oldAuthorities, verbose)
    while(diffW > Eps):
        if verbose:
            print "(Iteration %d) : %f diff" % (numIteration, diffW)
        doIteration(authorities, hubs, graph, users, verbose)
        diffW = diff(authorities, oldAuthorities, verbose)
    return authorities
### Intelligent Surfer PageRank
### ispr.py

```python
import snap
import csv
import time
import os
import sys
import string
import math

# Debugging

gDebug = False

def LoadGraph(pFilename):
    # Initialization
    vGraphTrain = snap.TNEANet.New()
    vGraphAll = snap.TNEANet.New()
    vNodeTags = {}
    # [Node] = [List of tags]
    vTestQuesVotes = {}
    # [QuestionId][AnswererId] = net votes
    vTestQuesTags = {}
    # [QuestionId] = [List of tags]

    with open(pFilename, 'r') as f:
        vReader = csv.reader(f, dialect='excel')
        for vLine in vReader:
            # Load graph data
            ReadEdge(vGraphAll, vLine)
            if int(vLine[2]) % 10 == 0:
                # Testing graph
                # Index test questions: net votes
                vNetVotes = float(vLine[3]) - float(vLine[4])
                if int(vLine[2]) not in vTestQuesVotes:
                    vTestQuesVotes[int(vLine[2])] = {}
                    vTestQuesVotes[int(vLine[2])][int(vLine[1])] = vNetVotes
                # Index test questions: tags
                if int(vLine[2]) not in vTestQuesTags:
                    vTestQuesTags[int(vLine[2])] = []
                    vTestQuesTags[int(vLine[2])].extend(vLine[7].split(','))
            else:
                # Training graph
                ReadEdge(vGraphTrain, vLine)
                # Index node tags
                if int(vLine[0]) not in vNodeTags:
                    vNodeTags[int(vLine[0])] = []
                if int(vLine[1]) not in vNodeTags:
                    vNodeTags[int(vLine[1])] = []
                vNodeTags[int(vLine[0])].extend(vLine[7].split(','))
                vNodeTags[int(vLine[1])].extend(vLine[7].split(','))

    # Format indices
    for vNode in vNodeTags:
        vNodeTags[vNode] = list(set(vNodeTags[vNode]))
    for vQuestion in vTestQuesTags:
        vTestQuesTags[vQuestion] = list(set(vTestQuesTags[vQuestion]))

    return (vGraphTrain, vGraphAll, vNodeTags, vTestQuesVotes, vTestQuesTags)

def ReadEdge(pGraph, pLine):
    if pGraph.IsNode(int(pLine[0])) == False:
        pGraph.AddNode(int(pLine[0]))
    if pGraph.IsNode(int(pLine[1])) == False:
        pGraph.AddNode(int(pLine[1]))
    pGraph.AddEdge(int(pLine[0]), int(pLine[1]))
    vEdgeId = pGraph.GetEId(int(pLine[0]), int(pLine[1]))
    pGraph.AddIntAttrDatE(vEdgeId, int(pLine[2]), "QuestionId")
    pGraph.AddIntAttrDatE(vEdgeId, int(pLine[2]), "UpVotes")
    pGraph.AddIntAttrDatE(vEdgeId, int(pLine[2]), "DownVotes")
    pGraph.AddIntAttrDatE(vEdgeId, float(pLine[5]), "SrcRep")
    pGraph.AddIntAttrDatE(vEdgeId, float(pLine[6]), "DstRep")

def WritePagerank(pPR, pFilename):
    with open(pFilename, 'w') as f:
        for item in pPR:
            vNode = item.GetKey()
            vPageRank = item.GetDat()
            f.write(str(vNode)+', '+str(vPageRank)+'
')

def TagNode(pNodeTags):
    vTagNode = {}  # [Tag] = [List of nodes]
    for vNode in pNodeTags:
        for vTag in pNodeTags[vNode]:
            if vTag not in vTagNode:
                vTagNode[vTag] = []
            vTagNode[vTag].append(vNode)
    return vTagNode

def CalcIDF(pTagNodes, pNumQuestions):
    vTagIDF = {}  # [Tag] = Inverse document frequency
    for vTag in pTagNodes:
        vTagIDF[vTag] = math.log(float(pNumQuestions)/len(pTagNodes[vTag]))
    return vTagIDF
```

21
```python
def CopyGraph(pGraph, pNodeTags, pTag):
    vSubgraphNodes = snap.TIntV()
    for vNode in pNodeTags[pTag]:
        vSubgraphNodes.Add(vNode)
    vGraph = snap.GetSubGraph(pGraph, vSubgraphNodes)
    return vGraph

def ReadISPR(pISPRDir):
    vTagPR = {}
    vFileList = [f for f in os.listdir(pISPRDir)]
    for vFilename in vFileList:
        if vFilename != 'ispr_tags_stats.txt':
            vTag = string.replace(string.replace(vFilename, 'ispr', ''), '.txt', '')
            vReader = csv.reader(open(pISPRDir+vFilename, 'r'), dialect='excel')
            for vLine in vReader:
                vTagPR[vTag][int(vLine[0])] = float(vLine[1])
    return vTagPR

def PairwiseRank(pCutoff, pTagPR, pTestQuesVotes, pTestQuesTags, pTagIDF, pOption):
    vNumWrong = 0
    vNumRight = 0
    vCount = 0
    for vQuesId in pTestQuesVotes:
        if gDebug:
            print('vQuesId : %d (%d, %d) pct done' % (vQuesId, float(vCount) / len(pTestQuesVotes) * 100))
        if gDebug:
            print('Before:', len(pTestQuesVotes[vQuesId]))
        pTestQuesVotes[vQuesId] = {key: value for key, value in pTestQuesVotes[vQuesId].items() if value >= pCutoff}
        if gDebug:
            print('After:', len(pTestQuesVotes[vQuesId]))
        vQuestionSort = sorted(pTestQuesVotes[vQuesId], key=lambda l: pTestQuesVotes[vQuesId][l], reverse=True)
        if len(vQuestionSort) < 1:
            continue
        if gDebug:
            for vUserId in vQuestionSort:
                print('vUserId : %d' % vUserId)
        vISPR = QuestionISPR(vQuestionSort, pTestQuesTags[vQuesId], pTagPR, pTagIDF)
        if gDebug:
            for vUserId in vQuestionSort:
                print('vISPR[vUserId] : %f' % vISPR[vUserId])
        vQuesRight = 0
        vQuesWrong = 0
        if pOption == 'Top':
            vGood = True
            for vRank in range(1, len(vQuestionSort)):
                if vISPR[vQuestionSort[vRank]] >= vISPR[vQuestionSort[0]]:
                    vGood = False
                    break
            if vGood:
                vNumRight += 1
                vQuesRight += 1
            else:
                vNumWrong += 1
                vQuesWrong += 1
        if gDebug:
            print('vQuesWrong : %d, vQuesRight : %d' % (vQuesWrong, vQuesRight))
        elif pOption == 'All':
            for vRankLo in range(1, len(vQuestionSort)):
                for vRankHi in range(0, vRankLo):
                    if vISPR[vQuestionSort[vRankHi]] >= vISPR[vQuestionSort[vRankLo]]:
                        vNumRight += 1
                        vQuesRight += 1
                    else:
                        vNumWrong += 1
                        vQuesWrong += 1
        if gDebug:
            print('vQuesWrong : %d, vQuesRight : %d' % (vQuesWrong, vQuesRight))
    return (vNumWrong, vNumRight)

def QuestionISPR(pUsers, pTags, pTagPR, pTagIDF):
    vISPR = {}
    vMissing = {}  # User ID is Intelligent Surfer Pagerank
    vUser = snap.TIntV()
    vUser = snap.GetSubGraph(pGraph, vSubgraphNodes)
    return vGraph
```
for vTag in pTags:
    for vUser in pUsers:
        vISPR[vUser] += pTagPR[vTag][vUser]
        vMissing[vUser] -= 1
for vUser in pUsers:
    vISPR[vUser] += vMissing[vUser] * float(pTagPR['all'][vUser])

## Method 2: P(q) is TF-IDF (TF is binary, so it’s ignored)
for vTag in pTags:
    if vTag != 'python':
        vIDF = pTagIDF[vTag]
    else:
        vIDF = 0
    for vUser in pUsers:
        if vUser in pTagPR[vTag]:
            vISPR[vUser] += pTagPR[vTag][vUser] * vIDF

if gDebug:
    for vTag in pTags:
        print('pTagIDF[%s] : %f' % (vTag, pTagIDF[vTag]))
return vISPR

def main():
    vStartTime = time.time()
    vTrain = False
    if len(sys.argv) == 2:
        if str.upper(sys.argv[1]) == 'TRAIN':
            vTrain = True
    # # Read edge list into training and test graphs
    vFilename = './data/network_python.graph'
    print('Loading graph : ', vFilename)
    [vGraphTrain, vGraphAll, vNodeTags, vTestQuesVotes, vTestQuesTags] = LoadGraph(vFilename)
    print('Training graph : %d nodes, %d edges' % (vGraphTrain.GetNodes(), vGraphTrain.GetEdges()))
    print('Entire graph : %d nodes, %d edges' % (vGraphAll.GetNodes(), vGraphAll.GetEdges()))
    print('# of test questions : %d' % (len(vTestQuesVotes)))
    print('**Runtime : ', time.time() - vStartTime, ' seconds')
    if vTrain:
        print('Train the intelligent surfer!'
        # Initialize output folder
        vFilename = './data/ispr/'
        print('Remove all files in ', vFilename)
        filelist = [f for f in os.listdir(vFilename) ]
        for f in filelist:
            os.remove(vFilename + f)
        print('**Runtime : ', time.time() - vStartTime, ' seconds'
        # Calculate statistics
        vFilename = './data/ispr/ispr_tagsstats.txt'
        print('Tag statistics : ', vFilename)
        vTagNodes = TagNode(vNodeTags)
        vTagIDF = CalcIDF(vTagNodes, vGraphAll.GetNodes())
        with open(vFilename, 'w') as f:
            for vTag in vTagNodes:
                f.write(str(vTag) + ', ' + str(len(vTagNodes[vTag])) + ', ' + str(vTagNodes[vTag]) + 'n')
        print('**Runtime : ', time.time() - vStartTime, ' seconds'
        # Pagerank for entire training graph
        vFilename = './data/ispr/ispr_all.txt'
        print('Pagerank for all tags : ', vFilename)
        vPR = snap.TIntFltH()
        snap.GetPageRank(vGraphAll, vPR)
        WritePagerank(vPR, vFilename)
        print('**Runtime : ', time.time() - vStartTime, ' seconds'
        # Pagerank for each tag in training graph
        print('Calculating Pagerank for each tag'
        for vTag in vTagNodes:
            vGraphTag = CopyGraph(vGraphTrain, vTagNodes, vTag)
            if vGraphTag.GetNodes() > 0:
                vFilename = './data/ispr/ispr_' + vTag + '.txt'
                print('Pagerank : ', vFilename)
                vPR = snap.TIntFltH()
                snap.GetPageRank(vGraphTag, vPR)
                WritePagerank(vPR, vFilename)
                print('**Runtime : ', time.time() - vStartTime, ' seconds'
        # Pagerank for each node in testing graph
        print('Calculating ISPR for each question'
        # Initialization
        if not vTrain:
            print('Read data'
            vTagNodes = TagNode(vNodeTags)
            vTagIDF = CalcIDF(vTagNodes, vGraphAll.GetNodes())
            vISPRDir = './data/ispr/
            vTagPR = ReadISR(vISPRDir)
            vNumWrongTop = []  # [Cutoff] = number of wrong rankings for top answer
            vNumWrongAll = []  # [Cutoff] = number of wrong rankings for all answers
            vNumRightTop = []  # [Cutoff] = number of right rankings for top answer
            vNumRightAll = []  # [Cutoff] = number of right rankings for all answers
            # Computation
            for vCutoff in range(-5, 100):
                print('=', 80
                print('**Runtime : ', time.time() - vStartTime, ' seconds'
                print('Cutoff =', vCutoff)
                vNumWrongTop[vCutoff], vNumWrongAll[vCutoff], vNumRightTop[vCutoff], vNumRightAll[vCutoff] = PairwiseRank(vCutoff, vTagPR, vTestQuesVotes, 0.80)
vTestQuesTags, vTagIDF, 'Top')
[vNumWrongAll[vCutoff], vNumRightAll[vCutoff]] = PairwiseRank(vCutoff, vTagPR, vTestQuesVotes,
vTestQuesTags, vTagIDF, 'All')

# Output results
print '==' * 80
print 'Final output'
print '{0:8}{1:16} {2:16} {3:16} {4:16}'.format('Cutoff', 'vNumWrongTop', 'vNumRightTop', 'vNumWrongAll', 'vNumRightAll')
for vCutoff in range(-5, 100):
    print '{0:8d}{1:16d} {2:16d} {3:16d} {4:16d}'.format(vCutoff, vNumWrongTop[vCutoff], vNumRightTop[vCutoff], vNumWrongAll[vCutoff], vNumRightAll[vCutoff])
print '==Runtime: {0:1.3f} seconds'.format(vStartTime - time.clock())

if __name__ == '__main__':
    main()
Validating Algorithms

# tests efficiency from graph passed in as first parameter
# uses ~/.data/network_python.graph" if none passed in
# @author rjalco

import snap
import prw3
import csv
from math import log

FILE_NETWORK = "../data/network_python.graph"
FILE_PR = "../data/pagerank_net.txt"
FILE_HITS = "../data/hits_net.txt"

# LOAD FILES
#
# def loadFile(CUT_OFF_RANK):
# global soHashMap
# global prHashMap
# global hitsHashMap
# global questions
# questions = {}
# weights = {}
# soReputations = {}
# prReputations = {}
# hitsReputations = {}
# graph = snap.TNEANet.New()
#
# file = open(FILE_NETWORK, 'rU')
# reader = csv.reader(file, dialect='excel')
# for list in reader:
# srcNId = int(list[0])
# dstNId = int(list[1])
# eAttrQId = int(list[2])
# eAttrUpVotes = float(list[3])
# eAttrDownVotes = float(list[4])
# eAttrSrcRep = float(list[5])
# eAttrDstRep = float(list[6])
# eAttrTags = list[7]
# key = (srcNId, dstNId)
# if not key in weights:
# weights[key] = 0
# weights[key] = weights[key] + eAttrUpVotes - eAttrDownVotes
eNet = eAttrUpVotes - eAttrDownVotes
# if eAttrQId in questions:
# rank = float(list[1])
# prReputations[id] = rank
# file.close()
#
# file = open(FILE_PR, 'rU')
# reader = csv.reader(file, dialect='excel')
# next(reader, None)
# for list in reader:
# id = int(list[0])
# if not id in soReputations:
# continue
# rank = float(list[1])
# prReputations[id] = rank
# file.close()
#
# file = open(FILE_HITS, 'rU')
# reader = csv.reader(file, dialect='excel')
# next(reader, None)
# for list in reader:
# id = int(list[0])
# if not id in soReputations:
# continue
# rank = float(list[1])
# hitsReputations[id] = rank
# file.close()

sortedSO = sorted(soReputations, key=lambda l: soReputations[l], reverse=True)
sortedPR = sorted(prReputations, key=lambda l: prReputations[l], reverse=True)
sortedHits = sorted(hitsReputations, key=lambda l: hitsReputations[l], reverse=True)

soHashMap = {} for i in xrange(len(sortedSO)):
soHashMap[sortedSO[i]] = i
prHashMap = {} for i in xrange(len(sortedPR)):
prHashMap[sortedPR[i]] = i
hitsHashMap = {} for i in xrange(len(sortedHits)):
hitsHashMap[sortedHits[i]] = i

# # # HOW RANKINGS FARED
# # #
# def numRightTopAnswers(rankingHashMap):

# # # NEW RANKINGS FARED

# def numRightTopAnswers(rankingHashMap):
```python
def numWrongRankings(rankingHashMap):
    numWrong = 0
    numRight = 0
    for question in questions.keys():
        questionSort = sorted(questions[question], key=lambda i: questions[question][1], reverse=True)
        good = True
        if len(questionSort) <= 1:
            continue
        for i in range(1, len(questionSort)):
            if not questionSort[i] in rankingHashMap:
                continue
            if rankingHashMap[questionSort[i]] < rankingHashMap[questionSort[0]]:
                good = False
                break
            if good:
                numRight += 1
            else:
                numWrong += 1
    return (numWrong, numRight)

def numRightRankings(rankingHashMap):
    numWrong = 0
    numRight = 0
    for question in questions.keys():
        questionSort = sorted(questions[question], key=lambda i: questions[question][1], reverse=True)
        good = True
        if len(questionSort) <= 1:
            continue
        for i in range(len(questionSort)):
            for j in range(i):
                if not questionSort[i] in rankingHashMap:
                    continue
                if rankingHashMap[questionSort[i]] <= rankingHashMap[questionSort[j]]:
                    numRight += 1
                else:
                    numWrong += 1
    return (numWrong, numRight)
```

```
Ingestion from Stack Overflow

/*
-- Number of posts per day
WITH tblPosts (Id, CreationDate) AS
(
  SELECT a.Id,
        CAST(a.CreationDate AS DATE)
  FROM Posts a
  WHERE a.CreationDate BETWEEN '2008-08-01' AND '2008-08-10'
)
SELECT a.CreationDate,
       COUNT(a.Id)
FROM tblPosts a
GROUP BY a.CreationDate
ORDER BY 1;
*/

/*
-- Number of each PostType per day
WITH tblPosts (Id, CreationDate, PostType) AS
(
  SELECT a.Id,
        CAST(a.CreationDate AS DATE),
        b.Name
  FROM Posts a,
       PostTypes b
  WHERE a.PostTypeId = b.Id
)
SELECT a.CreationDate,
       a.PostType,
       COUNT(a.Id)
FROM tblPosts a
GROUP BY a.CreationDate, a.PostType
ORDER BY 1, 2;
*/

/*
-- Mimic the following post:
-- http://stackoverflow.com/questions/613183/python-sort-a-dictionary-by-value
SELECT b.Name AS PostType,
       a.Score AS PostScore,
       a.Tags AS PostTags,
       a.Body AS PostBody,
       c.Score AS CommentScore,
       c.Text AS CommentText
FROM Posts a,
     PostTypes b,
     Comments c
WHERE a.PostTypeId = b.Id
AND a.Id = c.PostId
AND (a.Id = 613183 OR a.ParentId = 613183)
ORDER BY b.Name DESC, a.Score DESC, c.Score DESC;
*/

-- Votes on Question
SELECT b.Name,
       COUNT(1)
FROM Votes a,
     VoteTypes b
WHERE a.VoteTypeId = b.Id
AND a.PostId = 613183
AND (a.Id = 613183 OR a.ParentId = 613183)
GROUP BY b.Name;

-- Votes on Answers
WITH tblVoteAnswer (PostId, PostScore, VoteType, NumVotes) AS
(
  SELECT c.Id,
        c.Score,
        c.Posts,
        c.PostTypes d
  WHERE a.VoteTypeId = b.Id
  AND a.PostId = c.Id
  AND c.PostId = d.Id
  AND c.ParentId = 613183
  AND d.Name = 'Answer'
  GROUP BY b.Name, c.Id, c.Score
)
SELECT a.PostId,
       COUNT(1) AS Score
FROM
(
  SELECT PostId, NumVotes
  FROM tblVoteAnswer
  WHERE VoteType = 'UpMod'
) a
LEFT JOIN
(
  SELECT PostId, NumVotes
  FROM tblVoteAnswer
  WHERE VoteType = 'DownMod'
) b
ON a.PostId = b.PostId

27
Stack Overflow does not provide the user ID of up/down votes.

```sql
SELECT * FROM Votes WHERE PostId = 613218;  -- answer
```

With the `tblVoteAnswer` function:

```sql
WITH tblVoteAnswer (PostId, PostScore, OwnerUserId, VoteType, NumVotes) AS (
    SELECT c.Id, c.Score, c.OwnerUserId, b.Name, COUNT(*)
    FROM Votes a, VoteTypes b, Posts c, PostTypes d
    WHERE a.VoteTypeId = b.Id
    AND a.PostId = c.Id
    AND c.PostTypeId = d.Id
    AND c.ParentId = 613183
    AND d.Name = 'Answer'
    GROUP BY b.Name, c.Id, c.Score, c.OwnerUserId
)
SELECT a.PostId, a.OwnerUserId, a.NumVotes - COALESCE(b.NumVotes, 0) AS Score,
CASE WHEN (a.NumVotes + COALESCE(b.NumVotes, 0)) > 0
    THEN CAST(a.NumVotes AS DOUBLE PRECISION) / (a.NumVotes + COALESCE(b.NumVotes, 0))
    ELSE 0
END AS PosEval
FROM tblVoteAnswer a LEFT JOIN tblVoteAnswer b
ON a.PostId = b.PostId
ORDER BY Score DESC;
```

User IDs and names of people in Question 613183:

```sql
SELECT CASE WHEN b.Name = 'Question'
    THEN a.Id
    ELSE a.ParentId
END AS QuestionId, b.Name AS PostType, a.Score, a.Tags, a.OwnerUserId, c.DisplayName
FROM Posts a, PostTypes b, Users c
WHERE a.PostTypeId = b.Id
AND a.OwnerUserId = c.Id
AND (a.Id = 613183 OR a.ParentId = 613183)
ORDER BY b.Name DESC, a.Score DESC;
```

Network of people in Question 613183:

```sql
WITH tblQuestionUsers (QuestionId, PostType, Score, Tags, OwnerUserId, DisplayName) AS (
    SELECT CASE WHEN b.Name = 'Question'
    THEN a.Id
    ELSE a.ParentId
    END AS QuestionId, b.Name AS PostType, a.Score, a.Tags, a.OwnerUserId, c.DisplayName
FROM Posts a, PostTypes b, Users c
WHERE a.PostTypeId = b.Id
AND a.OwnerUserId = c.Id
AND (a.Id = 613183 OR a.ParentId = 613183)
)
SELECT a.QuestionId, a.OwnerUserId AS SrcNodeId, b.OwnerUserId AS DstNodeId, a.Score AS TotalVote
FROM (
    SELECT a.QuestionId, a.OwnerUserId, a.Score
    FROM tblQuestionUsers a
    WHERE a.PostType = 'Answer'
) AS a,
( SELECT a.QuestionId, a.OwnerUserId, a.Score
FROM tblQuestionUsers a
WHERE a.PostType = 'Question'
) AS b
ORDER BY 2 DESC;
```
WHERE a.QuestionId = b.QuestionId
ORDER BY TotalVote DESC;

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

SELECT CAST('Answer' AS NVARCHAR(50)),
c.ParentId AS QuestionId,
c.Id AS AnswerId,
b.Name AS VoteType,
COUNT(*) AS NumVotes
FROM Votes a,
     VoteTypes b,
     Posts c,
     PostTypes d
WHERE a.VoteTypeId = b.Id
AND a.PostId = c.Id
AND c.PostTypeId = d.Id
AND c.ParentId = 613183
AND d.Name = 'Answer'
GROUP BY c.ParentId, c.Id, b.Name
UNION ALL

SELECT 'Question',
a.PostId AS QuestionId,
a.PostId
b.Name AS VoteType,
COUNT(*) AS NumVotes
FROM Votes a,
     VoteTypes b
WHERE a.VoteTypeId = b.Id
AND a.PostId = 613183
GROUP BY a.PostId, b.Name

SELECT a.PostType,
a.QuestionId,
a.AnswerId,
a.NumVotes AS UpVote,
COALESCE(b.NumVotes, 0) AS DownVote,
a.NumVotes - COALESCE(b.NumVotes, 0) AS Score
FROM (
  SELECT *
  FROM tblVotes a
  WHERE a.VoteType = 'UpMod'
) a
LEFT JOIN
(
  SELECT *
  FROM tblVotes a
  WHERE a.VoteType = 'DownMod'
) b
ON a.AnswerId = b.AnswerId
ORDER BY PostType DESC, Score DESC;

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

SELECT a.TagName,
a.CreationYear,
a.CreationMonth,
COUNT(*) AS NumQuestions,
SUM(a.AnswerCount) AS SumAnswerCount,
AVG(CAST(a.AnswerCount AS DOUBLE PRECISION)) AS AvgAnswerCount,
STDDEV(CAST(a.AnswerCount AS DOUBLE PRECISION)) AS StdevAnswerCount
FROM (SELECT b.TagName,
         YEAR(c.CreationDate) AS CreationYear,
         MONTH(c.CreationDate) AS CreationMonth,
         c.*
         FROM PostTags a,
              Tags b,
              Posts c
         WHERE a.TagId = b.Id
         AND b.TagName IN ('python', 'c++', 'ios', 'sql', 'linux')
         AND a.PostId = c.Id
      ) a
GROUP BY a.TagName, a.CreationYear, a.CreationMonth
ORDER BY 1, 2, 3;

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

SELECT a.Id,
a.OwnerUserId,
SUM(CASE WHEN b.VoteTypeId = 2 THEN 1 ELSE 0 END) AS UpVotes,
SUM(CASE WHEN b.VoteTypeId = 3 THEN 1 ELSE 0 END) AS DownVotes,
a.Reputation
FROM tblOne a,
     tblTwo b
ON a.a = b.b
ORDER BY 1, 2, 3;

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

SELECT *
FROM PostTags a,
     Tags b,
     Posts c
WHERE a.TagId = b.Id
AND b.TagName IN ('python', 'c++', 'ios', 'sql', 'linux')
AND a.PostId = c.Id
ORDER BY 1, 2, 3;

SELECT *
FROM PostTags a,
     Tags b,
     Posts c
WHERE a.TagId = b.Id
AND b.TagName IN ('python', 'c++', 'ios', 'sql', 'linux')
AND a.PostId = c.Id
ORDER BY 1, 2, 3;

SELECT *
FROM PostTags a,
     Tags b,
     Posts c
WHERE a.TagId = b.Id
AND b.TagName IN ('python', 'c++', 'ios', 'sql', 'linux')
AND a.PostId = c.Id
ORDER BY 1, 2, 3;

SELECT *
FROM PostTags a,
     Tags b,
     Posts c
WHERE a.TagId = b.Id
AND b.TagName IN ('python', 'c++', 'ios', 'sql', 'linux')
AND a.PostId = c.Id
ORDER BY 1, 2, 3;

SELECT *
FROM PostTags a,
     Tags b,
     Posts c
WHERE a.TagId = b.Id
AND b.TagName IN ('python', 'c++', 'ios', 'sql', 'linux')
AND a.PostId = c.Id
ORDER BY 1, 2, 3;
FROM (SELECT a.*, e.Reputation FROM Posts a, PostTypes b, PostTags c, Tags d, Users e WHERE a.PostTypeId = b.Id AND a.Id = c.PostId AND c.TagId = d.Id AND a.OwnerUserId = e.Id AND b.Name = 'Question' AND d.TagName IN ('python') AND a.CreationDate BETWEEN '2009-03-01' AND '2009-03-31' ) a LEFT JOIN Votes b ON a.Id = b.PostId GROUP BY a.Id, a.OwnerUserId, a.Reputation ORDER BY 1;

-- Answers: up/down votes, reputation
SELECT a.Id, a.ParentId, a.OwnerUserId, SUM(CASE WHEN b.VoteTypeId = 2 THEN 1 ELSE 0 END) AS UpVotes, SUM(CASE WHEN b.VoteTypeId = 3 THEN 1 ELSE 0 END) AS DownVotes, a.Reputation FROM (SELECT a.*, f.Reputation FROM Posts a, PostTypes b, PostTags c, Tags d, Posts e, Users f WHERE a.PostTypeId = b.Id AND a.ParentId = c.PostId AND c.TagId = d.Id AND e.Id = a.ParentId AND a.OwnerUserId = f.Id AND b.Name = 'Answer' AND d.TagName IN ('python') AND e.CreationDate BETWEEN '2009-03-01' AND '2013-03-31' ) a LEFT JOIN Votes b ON a.Id = b.PostId GROUP BY a.Id, a.ParentId, a.OwnerUserId, a.Reputation ORDER BY 1;

-- Network
WITH tblQuestion (Id, OwnerUserId, UpVotes, DownVotes, Reputation) AS (SELECT a.Id, a.OwnerUserId, SUM(CASE WHEN b.VoteTypeId = 2 THEN 1 ELSE 0 END) AS UpVotes, SUM(CASE WHEN b.VoteTypeId = 3 THEN 1 ELSE 0 END) AS DownVotes, a.Reputation FROM (SELECT a.*, a.OwnerUserId, SUM(CASE WHEN b.VoteTypeId = 2 THEN 1 ELSE 0 END) AS UpVotes, SUM(CASE WHEN b.VoteTypeId = 3 THEN 1 ELSE 0 END) AS DownVotes, a.Reputation FROM Posts a, PostTypes b, PostTags c, Tags d, Users e WHERE a.PostTypeId = b.Id AND a.OwnerUserId = e.Id AND b.Name = 'Question' AND d.TagName IN ('python') AND a.CreationDate BETWEEN '2009-03-01' AND '2009-03-31' ) a LEFT JOIN Votes b ON a.Id = b.PostId WHERE a.Id = a.OwnerUserId, a.Reputation ORDER BY 1)
)

tblAnswer (Id, ParentId, OwnerUserId, UpVotes, DownVotes, Reputation) AS (SELECT a.Id, a.ParentId, a.OwnerUserId, SUM(CASE WHEN b.VoteTypeId = 2 THEN 1 ELSE 0 END) AS UpVotes, SUM(CASE WHEN b.VoteTypeId = 3 THEN 1 ELSE 0 END) AS DownVotes, a.Reputation FROM (SELECT a.*, a.OwnerUserId, SUM(CASE WHEN b.VoteTypeId = 2 THEN 1 ELSE 0 END) AS UpVotes, SUM(CASE WHEN b.VoteTypeId = 3 THEN 1 ELSE 0 END) AS DownVotes, a.Reputation FROM Posts a, PostTypes b, PostTags c, Tags d, Posts e, Users f WHERE a.PostTypeId = b.Id AND a.ParentId = c.PostId AND c.TagId = d.Id AND a.OwnerUserId = f.Id AND b.Name = 'Answer' AND d.TagName IN ('python') AND a.CreationDate BETWEEN '2009-03-01' AND '2009-03-31' ) a LEFT JOIN Votes b ON a.Id = b.PostId WHERE a.Id = a.OwnerUserId, a.Reputation ORDER BY 1)
AND b.Name = 'Answer'
AND d.TagName IN ('python')
AND e.CreationDate BETWEEN '2009-03-01' AND '2009-03-31'
) a LEFT JOIN
Votes b
ON a.Id = b.PostId
GROUP BY a.Id, a.ParentId, a.OwnerUserId, a.Reputation

SELECT a.QuestionId,
a.SrcNodeId,
a.DstNodeId,
a.EdgeAttrUpVotes,
a.EdgeAttrDownVotes,
a.EdgeAttrSrcRep,
a.EdgeAttrDstRep,
b.Tags AS EdgeAttrTags
FROM (SELECT a.Id AS QuestionId,
a.OwnerUserId AS SrcNodeId,
b.OwnerUserId AS DstNodeId,
b.UpVotes AS EdgeAttrUpVotes,
b.DownVotes AS EdgeAttrDownVotes,
a.Reputation AS EdgeAttrSrcRep,
b.Reputation AS EdgeAttrDstRep
FROM tblQuestion a,
tblAnswer b
WHERE a.Id = b.ParentId)
a,
b
WHERE a.QuestionId = b.Id
-- AND a.QuestionId = 613183 -- DEBUG 1
-- AND a.DstNodeId = 2786 -- DEBUG 2
ORDER BY a.SrcNodeId, a.DstNodeId;