1. What is your take away message from the paper?

The paper presents a couple strategies for recommending related functions given a single function, showing how with structural information guaranteed to be present good recommendations can be made using steady-state random walks over the API's callpath graph and association rule mining.

2. What is the motivation for the work (both people problem and technical problem), and its distillation into a research question? Why doesn't the people problem have a trivial solution? What are the previous solutions and why are they inadequate?

This paper is motivated by the people problem of developers spending a lot of time searching for information and the corresponding technical problem of how to reduce the time searching by using Recommendation Systems for Software Engineering (RSSEs). The problem is not trivial as it requires analysis of a lot of possible factors. Previously developers would have to dig through existing artifacts themselves, ask an expert, or perform multiple keyword searches. This is time-consuming. RSSE tools help users sift through the data and may even catch things that developers didn't know they were looking for.

Saul et al. tackle the specific question of whether a ranking system involving determining leveled neighborhoods in the callgraph centered around a specific function call can be improved with random walks and also consider a method using association rule mining.

3. What is the proposed solution (hypothesis, idea, design)? Why is it believed it will work? How does it represent an improvement? How is the solution achieved?

In general, the proposed solution to the problem is RSSEs, software that can detect what might be necessary for a developer and provide it automatically or at least provide better search in comparison to a general search engine with respect to the context of engineering. Saul et al.‘s solution compares itself to an existing Eclipse plug-in, Suade. The researchers hypothesize that using a larger set of possible recommendations built through a more extensive neighborhood search that is structured by levels will yield better recommendations. They further hypothesize that using steady-state random walk probabilities to rank possible recommendations will serve them well. They believe this is a better solution because developers tend to program at the same level of abstraction, hence a simple neighborhood will lose that context. They also see that similar ranking techniques have proven effective in general search (HITS) and build an analogy to their specific problem.

4. What is the author’s evaluation of the solution? What logic, argument, evidence, artifacts (e.g. a proof-of-concept system), or experiments are presented in support of the idea?

Saul et al. evaluate their work using basic metrics comparing their algorithms to Suade. From there they use case studies to demonstrate in what ways their system is superior. They also compare their recommendations to documentation as a form of verification. They use a metric of precision/recall (how many of their recommendations are useful and how many of the useful recommendations they found, respectively) to demonstrate which algorithms perform better overall, finding that a combination of their random walk and association rule recommenders works best. With the case studies they are able to show that their random walk predictor finds things that Suade cannot because of the callgraph neighborhood they use. They also demonstrate how their algorithm does better than a 'random recommender' though I find their claim that the random recommender emulates a new programmer suspect as the new programmer would have some English skill to help them in determining which function names are promising.
5. What is your analysis of the identified problem, idea and evaluation? Is this a good idea? What flaws do you perceive in the work? What are the most interesting and controversial ideas? For work that has practical implications, ask whether this will work, who would want it, what it will take to give it to them, and where might it become a reality?

I believe there are RSSE designs that will benefit developers. To get these to them, they must be developed and available as plug-in to IDEs and popular editors. It could also be quite useful in industry where code bases are proprietary and thus even search engines are not really an option. This could take the brunt off of experts and get new hires up to speed fast. It would be nice to see some user studies to see if developers will use these tools. While they have demonstrated good results by their definition, they have not shown that developers find them useful.

6. What are the paper's contributions (author's and your opinion)? Ideas, methods, software, experimental results, experimental techniques...?

Saul et al. contribute specific recommendation algorithms for the problem of finding related functions to a query function. I think one of their strongest elements is the targeting of functions on the same level of abstraction as the query and doing so by building the sibling-spouse enriched neighborhood graph. This is both a clever way of finding the abstraction layer and building in the context of how developers use an API. The authors also demonstrated the usefulness of random walks and association rule mining for this problem, and showed some difficulties in using them too, however I would have liked to have seen more of an explanation of the purpose of FRIAR. Also, there are no details regarding how expensive their algorithm is, especially in the data collection phase.

7. What are future directions for this research (authors' and yours, perhaps driven by shortcomings or critiques)?

Saul et al. note they have not yet evaluated their algorithms for a multi-function query or determined why they get better results when they increase k in the top-k recommendations they consider during their metrics. They also admit they have not done a study with experts and claim it is beyond their scope. They might also want to demonstrate their algorithm works on other code bases and build a plug-in for eclipse. Depending on how long the data collection phase takes, creating a way to gain the data regularly as the repository is updated by only rebuilding the parts that have changed would also be useful. I would have liked to have seen some study into what recommendations were missed (especially those that were not missed due to lower ranking).

8. What questions are you left with? What questions would you like to raise I an open discussion of the work (review interesting and controversial points)? What do you find difficult to understand? List as many as you can.

How do FRAN, FRIAR and CAR-B handle dead code and deprecated functions? How does the portability layer of Apache that was used in this example generalize? What about all of the cases in which FRAN does not generate more results or even generates less, what is happening there? How hard is this system to really set up and how long does it take, especially the parts that involve other existing tools like R and CodeSurfer? Why is the paper titled Recommending Random Walks when a large portion of it is not about the random walk part? Why frame it as infinite random walks instead of steady-state Markov chains? What kinds of functions does this system have no chance of picking up?

9. How do the concepts and approaches in this paper relate to your research project?

Saul et al. observe how methods relate through the callgraph in order to identify those that exist at the same level of abstraction and are likely related. In my project, I can similarly traverse the callgraph to group methods that likely belong to the same phase.