Recognising and Recommending Context in Social Web Search

Zurina Saaya, Barry Smyth, Maurice Coyle and Peter Briggs

CLARITY: Centre for Sensor Web Technologies
School of Computer Science and Informatics
University College Dublin, Ireland
firstname.lastname@ucd.ie
http://www.clarity-centre.org

Abstract. In this paper we focus on an approach to social search, HeyStaks that is designed to integrate with mainstream search engines such as Google, Yahoo and Bing. HeyStaks is motivated by the idea that Web search is an inherently social or collaborative activity. Heystaks users search as normal but benefit from collaboration features, allowing searchers to better organise and share their search experiences. Users can create and share repositories of search knowledge (so-called search staks) in order to benefit from the searches of friends and colleagues. As such search staks are community-based information resources. A key challenge for HeyStaks is predicting which search stak is most relevant to the users current search context and in this paper we focus on this so-called stak recommendation issue by looking at a number of different approaches to profling and recommending community-search knowledge.

Keywords: social search, context recommendation

1 Introduction

The *social web* is represented by a class of web sites and applications in which user participation is the primary driver of value. Discussions of the social web often use the phrase *collective intelligence* or *wisdom of crowds* to refer to the value created by the collective contributions of all these people writing articles for Wikipedia[1], sharing tagged photos on Flickr[2], sharing bookmarks on Delicious[3], streaming their personal blogs into the open seas of the blogosphere and using and sharing the search knowledge in collaborative environment[3].

Recently ideas from the social web has begun to exert their influence beyond content creation and on to content curation and information discovery. In short, many researchers have begun to consider the role of collaboration during information search and content discovery; see for example the work of Ariadne [10], SearchTogether[6] and CoSearch[1]. Golovchinsky et al. [2] proposes a taxonomy

[1] http://www.wikipedia.org
[2] http://www.flickr.com
[3] http://www.delicious.com

of collaborative information sharing highlight key dimensions such as the *intent, depth, concurrency,* and *location* for a variety of collaborative information services. Very briefly, for example, Golovchinsky et al. distinguish between services that support *implicit* versus *explicit* collaboration, services that over shallow UI-based collaboration versus deeper algorithmic support for collaboration, services that support *synchronous* versus *asynchronous* collaboration , and finally those services that assume information seekers are *co-located* versus those that assume *remote* collaboration.

In this paper we will focus on HeyStaks, the details of which have been previously published in [8]. In short, HeyStaks brings a layer of collaboration to mainstream search engines, via a browser plugin which allows searchers to organise and share their search experiences and to collaborate with others as they search. HeyStaks is a collaborative web search service that offers elements of implicit and explicit intent among searchers. It provides for a range of UI enhancements to support collaborating searchers as well as deeper algorithmic components in order to identify relevant results from a community of collaborators. Finally, it assumes asynchronous, remote collaboration: searchers do not need to be co-located and collaboration can occur overtime as recent searchers benefit from recommendations that originate from earlier search sessions.

Here we are emphasised on a key challenge for HeyStaks and its users. Specifically, the central concept in HeyStaks is the notion of a *search stak*, which acts like a folder for our search experiences. Briefly, a user can create a search stak on a topic of their choosing and they can opt to share this stak with other users. Now, as they search (using HeyStaks and their favourite mainstream search engine) the results that they select (or tag or share) will be associated with their active stak so that these results can be subsequently recommended to other stak members in the future when appropriate. In this way, stak members can benefit from the past searches of friends or colleagues who share their staks. A key problem here for HeyStaks to ensure that the right stak is chosen for a given search session. One way to solve this is to ask the user to pick their stak at the start of their search session, but since many users forget to do this, this is not a practical solution in reality. The alternative is to use information about the user's current search session as the basis for automatically selecting and recommending an appropriate stak at search time. In this paper then we focus on this stak selection problem and in what follows we describe and evaluate a recommendation-based strategy that works well enough in practice to automatically suggest relevant staks to the user at search time, or even automatically switch users into a likely stak without their intervention.

2 A Review of HeyStaks

In designing HeyStaks our primary goal is to provide social Web search enhancements, while at the same time allowing searchers to continue to use their favourite search engine. HeyStaks adds two basic features to any mainstream search engine. First, it allows users to create *search staks*, as a type of folder

for their search experiences at search time, and the creator can invite initial members by providing their email addresses. Staks can be configured to be *public* (anyone can join) or *private* (invitation only). Second, HeyStaks uses staks to generate recommendations that are added to the underlying search results that come from the mainstream search engine. These recommendations are results that stak members have previously found to be relevant for similar queries and help the searcher to discover results that friends or colleagues have found interesting, results that may otherwise be buried deep within Google's default result-list.

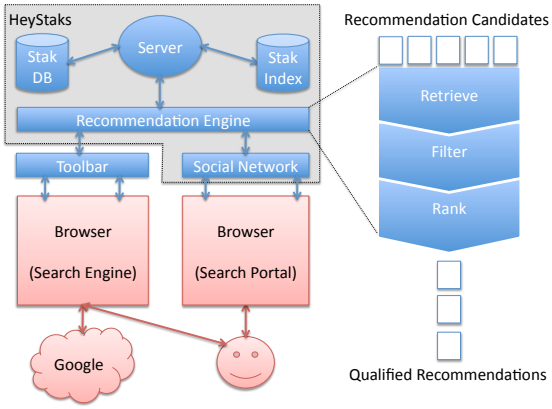

Fig. 1. The HeyStaks system architecture and outline recommendation model.

As shown in Figure 1, HeyStaks takes the form of two basic components: a client-side *browser toolbar* and a back-end *server*. The toolbar (see Figure 2) allows users to create and share staks and provides a range of ancillary services, such as the ability to tag or vote for pages. The toolbar also captures search result click-thrus and manages the integration of HeyStaks recommendations with the default result-list. The back-end server manages the individual stak indexes (indexing individual pages against query/tag terms and positive/negative votes), the stak database (stak titles, members, descriptions, status, etc.), the HeyStaks social networking service and, of course, the recommendation engine.

In the following sections we review how HeyStaks captures search activities within search staks and how this search knowledge is used to generate and filter result recommendations at search time; more detailed technical details can be found in [9, 8].

2.1 Profiling Stak Pages

Each stak in HeyStaks captures the search activities of its stak members within the stak's context. The basic unit of stak information is a result (URL) and

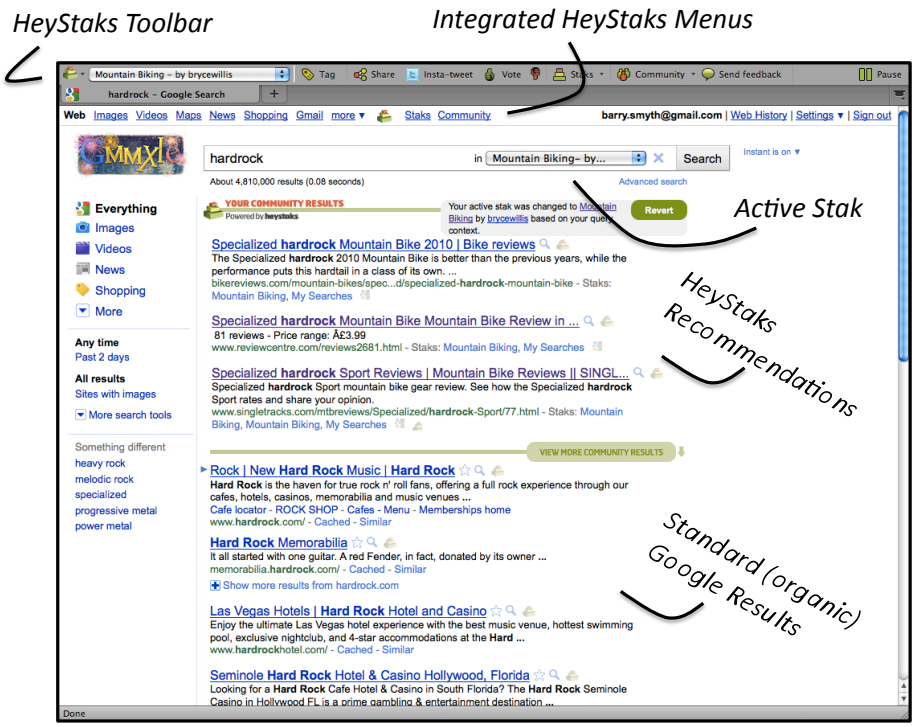

Fig. 2. The searcher is looking for information from a specialist mountain biking brand, Hard Rock, but Google responds with results related to the restaurant/hotel chain. HeyStaks recognises the query as relevant to the the searcher's*Mountain Biking* stak and presents a set of more relevant results drawn from this stak.

each stak (S) is associated with a set of results, $S = \{r_1, ..., r_k\}$. Each result is also anonymously associated with a number of implicit and explicit interest indicators, based on the type of actions that users can perform on these pages, which include:

- *Selections (or Click-thrus)* – that is, a user selects a search result (whether *organic* or *recommended*). Similarly,
  HeyStaks allows a user to *preview* a page by opening it in a frame (rather than a window), and to *popout* a page from a preview frame into a browser window;
- *Voting* – that is, a user positively votes on a given search result or the current web page;
- *Sharing* – that is, a user chooses to share a specific search result or web page with another user (via email or by posting to their Facebook wall etc.);
- *Tagging/Commenting* – that is, the user chooses to tag and/or comment on a particular result or web page.

Each of these actions can be associated with a degree of confidence that the user finds the page to be relevant for example, *implicit* actions such as result selections are weaker than *explicit* actions, such as tagging or sharing a page. Each result page r_i^S from stak S, is associated with these indicators of relevance, including the total number of times a result has been selected (Sl), the query terms $(q_1, ..., q_n)$ that led to its selection, the terms contained in the snippet of the selected result $(s_1, ..., s_j)$, the number of times a result has been tagged (Tg), the terms used to tag it $(t_1, ..., t_m)$, the votes it has received (v^+, v^-), and the number of people it has been shared with (Sh) as indicated by Equation 1.

$$r_i^S = \{q_1...q_n, s_1...s_j, t_1...t_m, v^+, v^-, Sl, Tg, Sh\} \ . \tag{1}$$

Importantly, this means each result page is associated with a set of *term data* (query terms and/or tag terms) and a set of *usage data* (the selection, tag, share, and voting count). The term data is represented as a Lucene (*lucene.apache.org*) index, with each result indexed under its associated query and tag terms, and this provides the basis for retrieving and ranking *recommendation candidates*. The usage data provides an additional source of evidence that can be used to filter results and to generate a final set of recommendations.

2.2 Recommending Results: Relevance & Reputation

At search time, the searcher's query q_T and current stak S_T are used to generate a list of recommendations to be returned to the searcher. There are two key steps when it comes to generating recommendations. First, a set of *recommendation candidates* are retrieved from S_T by querying the corresponding Lucene index with q_T. This effectively produces a list of recommendations based on the overlap between the query terms and the terms used to index each recommendation (query, snippet, and tag terms). Second, these recommendations are filtered and ranked. Results that do not exceed certain activity thresholds are eliminated as candidates; e.g., results with only a single selection or results with more negative votes than positive votes (see [8]). The remaining recommendation candidates are then ranked according to two key factors: *relevance* and *reputation*. Essentially each result is evaluated using a weighted score of its relevance and reputation score as per Equation 2; where w is used to adjust the relative influence of relevance and reputation and is usually set to 0.5.

$$score(r, q_T) = w \times rep(r) + (1 - w) \times rel(q_T, r) \ . \tag{2}$$

The relevance of a result r with respect to a query q_T is computed based on Lucene's standard *TF*IDF* metric [4] as per Equation 2. The reputation of a result is a function of the reputation of the stak members who have added the result to the stak. And their reputation in turn is based on the degree to which results that they have added to staks have been subsequently recommended to, and selected, by other users; see [5] for additional information.

3 Recognising Context & Recommending Staks

In this paper we are not concerned with recommending individual result pages to HeyStaks users. Rather, our focus is on the so-called *stak selection* task. Briefly, the success of HeyStaks depends critically on users correctly identifying an appropriate stak for their searches at search time. As in the example in Fig. 2, as the user search for mountain bike related information they need to choose *Mountain Biking* as their current stak. If they do this consistently then HeyStaks will learn to associate the right pages with the right staks, and be in a position to make high quality recommendations for stak members. However, the need to manually select a stak at the start of a new search session is an extra burden on the searcher. To make this as easy as possible, HeyStaks integrates its stak-lists as part of the mainstream search engine interface (see Fig. 2) but still many users, especially during the early stages forget to do this, and this means that a majority of search sessions are associated with the searcher's default stak (*My Searches*) rather than a more specific and appropriate stak of which they are a member.

The solution to this problem, which is the main contribution of this paper, is to proactively predict and recommend a suitable stak to the user at search time. To do this we draw on ideas from recommender systems and traditional information retrieval. As described above, each stak is a separate search index that is made up of documents that have been selected, tagged, and/or shared by stak members. For our stak recommendation solution we treat each stak index itself as a type of *summary* document; effectively the terms and URLs contained in the stak index become the terms of the summary document and in this way a collection of staks can be represented as a collection of documents. Using Lucene, these documents can then be transformed into a *stak summary index* (or *SSI*); see Fig. 3. Then, at search time, we can use the searcher's query as a probe into this stak summary index to identify a set of staks that most relevant to the query; in this work we focus only on staks that the user is currently a member of but a similar technique could be used to recommend other third-party staks in certain circumstances. These recommended staks can then be suggested directly to the user as a reminder to set their appropriate stak context; or, alternatively, we can configure HeyStaks to automatically pre-select the top ranking recommendation as the current stak context, while providing the searcher with an option to undo this if they deem the stak to be incorrect.

In the above we assume that the user's own search query (q_T) is used as the SSI query (or *stak query*), but in fact there are a number of additional sources of information that can be usefully harnessed for this. For example, at search time, the initial set of search engine results represents a valuable source of additional context information. This approach also has been used in [7] to classify the queries with text classification algorithm.

For instance, the terms in the title and snippets (R_{S+T}), and URLs (R_{URL}) of the result-list can also be used in addition to the user's short search query, during stak recommendation. For this reason we refer to three basic *types* of stak

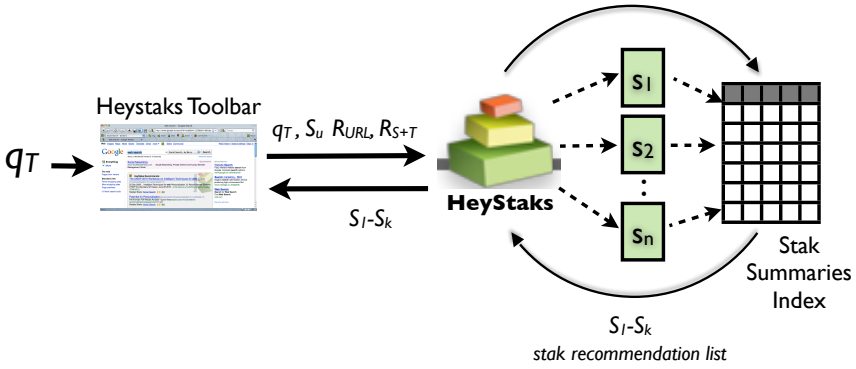

Fig. 3. Stak Recommendation.

recommendation strategy – *query, snippet, URL* – depending on which sources of information form the user's stak query (S_Q).

At stak recommendation time we use Lucene's standard TF*IDF weighting model as the basis for scoring recommended staks as shown in Equations 3 and 4. Effectively, terms in the stak summary index (SSI) are scored based on the TF*IDF model, which prefers terms that are frequent within a given stak but infrequent across the user's staks (S_U) as a whole.

$$RecList(S_Q, S_U, SSI) = \underset{\forall S \epsilon S_U}{SortDesc(Score(S_Q, S, SSI))} \tag{3}$$

$$Score(S_U, S, SSI) = \sum_{t \epsilon S_U} tf(t, S) \times idf(t, SSI) \tag{4}$$

In this way we can generate different recommendation lists (RL_{URL}, RL_{query}, RL_{S+T},) by using different sources of data as the stak query (S_Q); for example, we can use the terms in result titles and snippets as the stak query, which will lead to staks being recommended because they contain lots of distinctive title and snippet terms. Of course we can also look to combine these different sources of query terms, for example, by ranking recommended staks according to their position across the recommendation lists produced by different sources of query terms. For instance, we can define the *rank score* of a given stak, across a set of recommendation lists, to be the sum of the positions of the stak in the different recommendation lists with a simple penalty assigned for lists that do not contain the stak as per Equations 5 and 6. The final recommendation list is then sorted in ascending order of the rank scores of recommended staks.

$$RankScore(s, RL_1 - RL_n) = \sum_{RL_i \epsilon RL_1 - RL_n} PositionScore(s, RL_i) \tag{5}$$

$$PositionScore(s, RL) = \begin{cases} Position(s, RL) \text{ if } s \epsilon RL; \\ Length(RL) + 1 \text{ otherwise.} \end{cases} \quad (6)$$

We have described a general purpose approach to stak recommendation, which accommodates different sources of query data, and provides for a flexible way to combine multiple recommendation lists to generate an ensemble recommendation list. The intuition of course is that by combining different sources of query data we will generate better recommendations, which we shall look at in the following evaluation.

4 Evaluation

In this section we evaluate the different forms of our stak recommendation approach, based on live-user search data, and focusing in particular on the overall recommendation accuracy of the different techniques, and combinations of techniques, across different stak types.

4.1 Setup

The data for this evaluation stems from HeyStaks usage logs generated during the period October 2008 - October 2009. The sample data used contains 114,109 individual, timestamped search activities. Each refers to a specific search query submitted by a particular user in a given stak context. For the purpose of this evaluation we limit our interest to only those activities that are associated with non-default search staks; this means that we focused on search sessions where the user did select a specific stak for their search. There are 8,100 of these activities across 158 unique users and, on average, users were members of 6.94 staks each. We also collect data on the size of each of these staks, based on the number of URLs they contain to categorise staks as either *small, medium, large* or *extra-large* as per Table 1

Table 1. Staks Categories

# URLs	Size	# Staks	% of Staks
1 - 10	Small	378	63%
11 - 100	Medium	178	30%
101 - 500	Large	31	5%
500+	X-Large	11	2%

For the purpose of this study we evaluate a range of different recommendation strategies based on our three basic techniques, namely, *query, snippet, URL* and including all combinations of these techniques. In addition we also evaluate a baseline *random* recommendation strategy, which suggests staks at random

from the user's stak-list. This leads to a total of eight different recommendation alternatives. To evaluate these alternatives, we generate a recommendation list for each of the 8,100 search instances and compute the percentage of times that the known active stak is recommended among the top k recommendations ($k = 1 - 5$).

4.2 Overall Recommendation Precision

To begin with we will look at the overall success rate across the different recommendation alternatives. This data is presented in Fig. 4 as a graph of success rate against recommendation-list size (k). Each recommendation technique is represented as an individual line-graph based on its success rate for the different values of k. For clarity we also present the mean average success rate across the different values of k in Fig. 5.

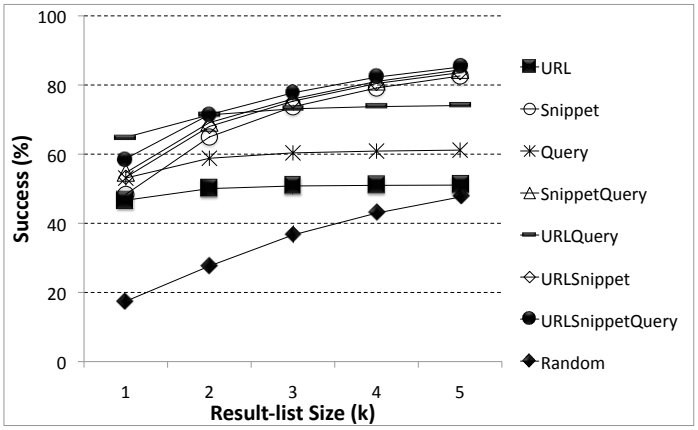

Fig. 4. Recommendation success rate

The results highlight a considerable variation in performance across the different recommendation strategies. As expected *random* performs poorly as the baseline, with a success rate of between 17 and 48% depending on k; as expected, success rates grow with increasing k since there are more opportunities to recommend the correct active stak. Generally speaking the ensemble approaches, which combine multiple basic techniques, tend to outperform individual techniques on their own. For example, one of the best performing strategies is the combination of URL, *snippet*, and *query* with success score ranging from 60% ($k = 1$) to 85% ($k = 5$), compared to the less impressive performance of say the URL technique on its own, which varies from about 47% ($k = 1$) to just 51% ($k = 5$).

It is interesting to pay special attention to the $k = 1$ results because the ideal strategy for HeyStaks would be to automatically switch the user into a

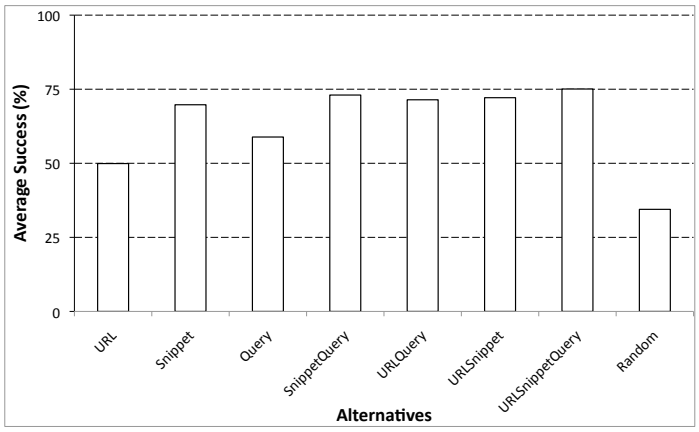

Fig. 5. Mean average success rate

correct stak, rather than present a set of stak options. This would require a reasonably high success rate at $k = 1$ to avoid user frustration in the case of incorrect stak switches. Unfortunately, it appears from the results in Fig. 4 that the success rates at $k = 1$ do not support such an automatic switching approach. For example, the best performing strategy at $k = 1$, which combines URL and *query* techniques, achieves a success rate of 65%, which does not seem high enough to support an automatic stak switching.

4.3 Precision vs Stak Size

Of course the above results refer to recommendation success across all staks. But not all staks are created equally. For example, as per Table 1, the majority of staks (63%) contain relatively few URLs (1-10 URLs) which provides a much weaker basis for indexing. It seems likely that this will have a great impact on stak recommendation effectiveness compared to larger staks. To test this, in Fig. 6 and Fig. 7 we present the recommendation success rate for each of the recommendation alternatives, by stak size (comparing *small, medium, large* and *extra-large* staks) for recommendation lists of size 1 (Fig. 6) and 3 (Fig. 7). It is clear that there are significant differences in recommendation accuracy across the various stak sizes. For example, looking at the combination of URL, *snippet*, *query* we see a success rate of about 75% at $k = 1$ for the extra-large staks and 70% for the large staks, compared to only 36% and 31% for the medium and small staks respectively. This is encouraging because, from a engineering standpoint, it suggests that it may be practical to implement a reliable automatic stak switching policy, at least for large staks which contain more than 100 URLs. When we look at the results for $k = 3$ (see Fig. 7) we see similar effects, only this time many ensemble techniques are achieving success rates in excess of 90%, for a number of recommendation combinations across the extra-large staks.

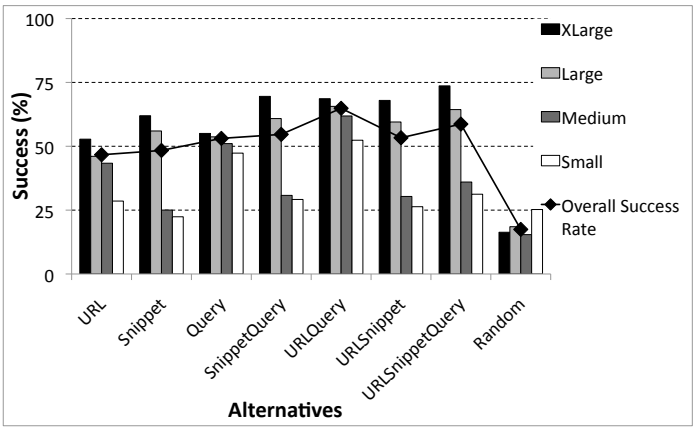

Fig. 6. Success rate by stak size where $k = 1$

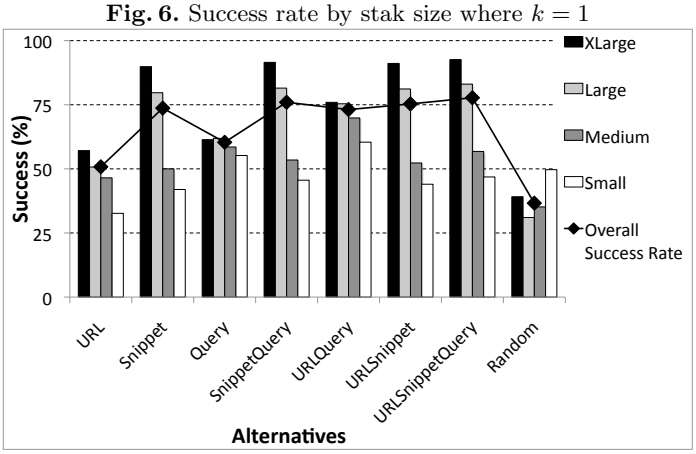

Fig. 7. Success rate by stak size where $k = 3$

4.4 Conclusions

HeyStaks facilitates partitioned collaboration between searchers, by allowing users to create and share their own search staks, and it does this by integrating with mainstream search engines rather than expecting the user to change to a new search engine. The main contribution of this work has been to highlight a practical problem facing HeyStaks — the need to automatically predict the right stak for users at search time — and to propose and evaluate a feasible solution in the form of a stak recommendation strategy. To this end we have described a general framework for stak recommendation. It is based on the indexing of user staks, which accommodates a variety of different recommendation alternatives using different types of query data at search time, such as search query terms, title and snippet terms of search results, the URLs of search results, and usage data from staks. We have described the results of a comprehensive evaluation of

a wide variety of recommendation strategies, based on live user search data, and the results speak to the practical effectiveness of this overall approach to stak recommendation. In particular, the success scores achieved across the larger staks speak to the potential for a reliable automatic stak switching mechanism, and at the very least it is possible to generate a short-list of stak recommendations that are accurate up to nearly 90% of the time.

Acknowledgments

This work is supported by Science Foundation Ireland under grant 07/CE/I1147, HeyStaks Technologies Ltd, Ministry of Higher Education Malaysia and Universiti Teknikal Malaysia Melaka.

References

1. S. Amershi and M. R. Morris. Cosearch: a system for co-located collaborative web search. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1647–1656, New York, NY, USA, 2008. ACM.
2. G. Golovchinsky, J. Pickens, and M. Back. A taxonomy of collaboration in online information seeking. In *JCDL Workshop on Collaborative Information Retrieval*, pages 1–3, 2008.
3. T. Gruber. Collective knowledge systems: Where the social web meets the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):4 – 13, 2008. Semantic Web and Web 2.0.
4. E. Hatcher and O. Gospodnetic. *Lucene in action*. Manning Publications, 2004.
5. K. McNally, M. P. O'Mahony, B. Smyth, M. Coyle, and P. Briggs. Towards a reputation-based model of social web search. In *IUI '10: Proceeding of the 14th international conference on Intelligent user interfaces*, pages 179–188, New York, NY, USA, 2010. ACM.
6. M. R. Morris and E. Horvitz. Searchtogether: an interface for collaborative web search. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 3–12, New York, NY, USA, 2007. ACM.
7. D. Shen, R. Pan, J.-T. Sun, J. J. Pan, K. Wu, J. Yin, and Q. Yang. Query enrichment for web-query classification. *ACM Trans. Inf. Syst.*, 24:320–352, July 2006.
8. B. Smyth, P. Briggs, M. Coyle, and M. O'Mahony. Google shared. a case-study in social search. In *Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization*, UMAP '09, pages 283–294, Berlin, Heidelberg, 2009. Springer-Verlag.
9. B. Smyth, P. Briggs, M. Coyle, and M. P. O'Mahony. A case-based perspective on social web search. In *Proceedings of the 8th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ICCBR '09, pages 494–508, Berlin, Heidelberg, 2009. Springer-Verlag.
10. M. B. Twidale, D. M. Nichols, and C. D. Paice. Browsing is a collaborative process. *Information Processing & Management*, 33(6):761 – 783, 1997.