SuggestBot: Using Intelligent Task Routing to Help People Find Work in Wikipedia

Dan Cosley
Cornell University
Human-Computer Interaction Group
drc44@cornell.edu

Dan Frankowski, Loren Terveen, John Riedl
University of Minnesota
GroupLens Research, CommunityLab*
{dfrankow, terveen, riedl}@cs.umn.edu

ABSTRACT
Member-maintained communities ask their users to perform tasks the community needs. From Slashdot, to IMDb, to Wikipedia, groups with diverse interests create community-maintained artifacts of lasting value (CALV) that support the group's main purpose and provide value to others. Said communities don't help members find work to do, or do so without regard to individual preferences, such as Slashdot assigning meta-moderation randomly. Yet social science theory suggests that reducing the cost and increasing the personal value of contribution would motivate members to participate more.

We present SuggestBot, software that performs intelligent task routing (matching people with tasks) in Wikipedia. SuggestBot uses broadly applicable strategies of text analysis, collaborative filtering, and hyperlink following to recommend tasks. SuggestBot's intelligent task routing increases the number of edits by roughly four times compared to suggesting random articles. Our contributions are: 1) demonstrating the value of intelligent task routing in a real deployment; 2) showing how to do intelligent task routing; and 3) sharing our experience of deploying a tool in Wikipedia, which offered both challenges and opportunities for research.

Keywords: online communities, member-maintained communities, Wikipedia, intelligent task routing, recommender systems

ACM Classification: H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces—Collaborative computing

Introduction

Why can't Dfrankow find articles to edit in Wikipedia? The opportunities are endless; the English version has over 1.2 million articles and the Community Portal page lists dozens of ways to contribute. Whether members are particular about punctuation, interested in images, or suckers for soccer, the portal can help them participate. New members might not be aware that the portal exists, but Dfrankow has made over 100 edits and is aware of it. He has diverse interests: music, the environment, computers, and vegetarian cooking. Yet he says it is hard to find articles that both need work and that he would like to work on.

These problems are not unique to Dfrankow, nor to Wikipedia. They arise in other *member-maintained communities* that solicit contributions from their users to perform tasks the community needs to function. Communities like Slashdot, Amazon, and digg ask members to evaluate the quality of other members' contributions, aggregating these judgements to perform *distributed moderation* [21]. Groups like Wikipedia and Distributed Proofreaders, a group of volunteers transcribing books into electronic forms for Project Gutenberg, welcome and mentor new members [32]. Many groups create *community-maintained artifacts of lasting value*, or CALVs [7], databases that support the group's main purpose. For example, rateyourmusic.com has created a database of over a million songs. The database is not the goal; rather, it is a tool that allows its members to make lists of songs and talk about them. IMDb's database of movies, the ChefMoz restaurant directory, and Wikipedia's articles are all built by members and provide value to millions of people. Many other communities build group-specific resources.

Most member-maintained communities don't help members find work. Few directly ask members to perform specific tasks; those that do fail to consider individuals' preferences when assigning tasks. Slashdot assigns meta-moderation randomly. Distributed Proofreaders categorizes texts into difficulty levels, but doesn't try to match people with appropriate tasks. Wikipedia provides dozens of ways to help, all of which are impersonal: randomly chosen and alphabetical lists of articles that need work, articles that many people would like to see created, and so on.

Yet social science theory suggests reducing the cost of contribution will increase members' motivation to participate. One way to do so is to make it easy to find work to do. Technologies such as information retrieval and recommender systems help people find valuable items to consume in a sea of possible choices, thus reducing their search costs. Communities might use these technologies to match members with appropriate work, increasing the ultimate value of the community.

* CommunityLab is a collaborative project of the University of Minnesota, University of Michigan, and Carnegie Mellon University. http://www.communitylab.org/

We gratefully acknowledge the support of the National Science Foundation, under grants IIS 03-24851 and IIS 05-34420.

IUI'07, January 28–31, 2007, Honolulu, Hawaii, USA.

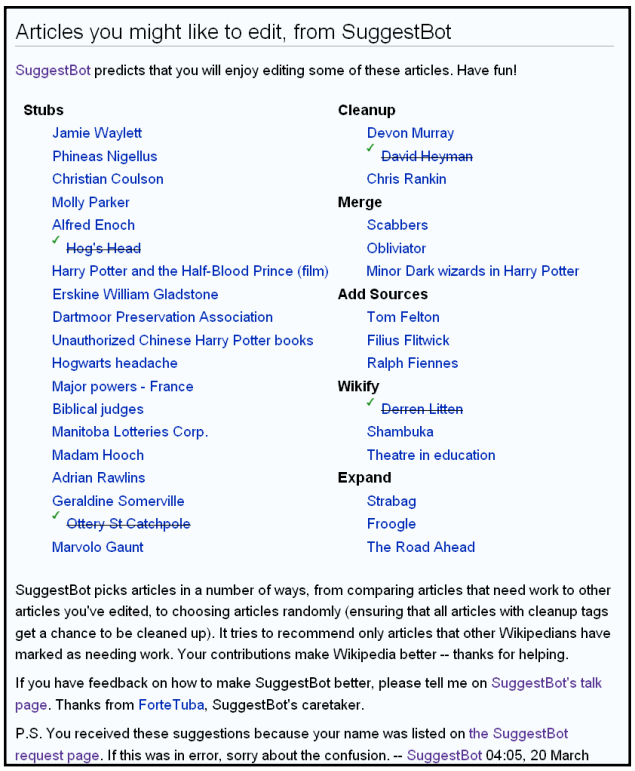

Figure 1: The current version of SuggestBot, making suggestions to Garykirk. The interface explains how and why he got these suggestions, and groups them by the type of work needed. He has edited and checked off several of the suggested items.

We call matching people with appropriate tasks *intelligent task routing*. In this paper, we describe SuggestBot, a system for intelligent task routing in Wikipedia. We first present theories and prior work that support the use of intelligent task routing as a mechanism for increasing contributions. We next describe how Wikipedia's extensive historical data and active community shaped the design of SuggestBot, followed by a description of how it works. Figure 1 depicts a page of suggestions for a Wikipedia user named Garykirk. To create the suggestions, it used Garykirk's history of edits as input to simple, general recommendation algorithms based on text matching, link following, and collaborative filtering. We close by evaluating SuggestBot. On average, the intelligent algorithms produce suggestions that people edit four times as often as randomly chosen articles. This large difference suggests that intelligent task routing is a powerful tool for encouraging contributions to communities.

Intelligent task routing, theory and practice

CommunityLab is a collaboration between the University of Michigan, the University of Minnesota, and Carnegie Mellon University. This project uses social science theories concerning motivation to design systems that increase participation in online communities. We have studied how contributions to groups are affected by goal-setting [1], knowing who benefits from a contribution [26, 34], the effect of similarity to other group members [9, 27], linguistic characteristics of discussion posts [18], the costs and benefits of contributing [13], and editorial review [6]. Here, we bring CommunityLab techniques to bear on the challenge of intelligent task routing. Both economics and social psychology suggest that reducing the cost of contributing will increase people's motivation to do work for a community.

Economics provides the theory of public goods [11]. Public goods have two special characteristics. First, once they are produced, anyone can use them. Second, they cannot be used up by any one person. National defense and public radio are examples. The theory states that groups generally produce less than the optimal amount of a public good because members have an incentive to use the work of others instead of making contributions. This *free riding* behavior has been demonstrated in a number of laboratory experiments [24]. Thorn and Connolly use public goods theory to model *discretionary databases*, which are similar to CALVs in that they are online, are shared by a group, and contributions to the database are voluntary [5]. Their work shows that, as public goods theory predicts, lowering the cost of contributing increases contributions to the database.

In social psychology, Karau and Williams' *collective effort model* addresses *social loafing*, the observation that people often contribute less toward group tasks than they would if they performed the same task individually [22]. The model integrates a number of explanations of social loafing (e.g., [12, 19]). It predicts that reducing costs and increasing the value of outcomes will increase motivation. Since intelligent task routing reduces the cost of finding work and matches people with tasks they are likely to care about, it should increase people's motivation to contribute.

We used the collective effort model to shape our first study of intelligent task routing [7]. The goal was to increase how often members corrected information in the MovieLens web site's database by matching members with movies they were likely to edit. The study compared four very simple strategies for choosing movies based on the collective effort model: choosing random movies, choosing movies the person was predicted to like, choosing movies that were missing the most information in the database, and choosing movies the person had previously rated. Choosing movies with high predictions did worse than random. Choosing movies that were missing the most information did better than random, but the winner by far was to choose rated movies. Four times as many people in that group edited at least one movie compared to the other groups.

We believe this is because people who had seen a movie already knew much of the pertinent information for MovieLens, such as actors, directors, and language. Knowing this information reduced the cost of making a contribution and, as theory predicted, increased people's motivation to contribute. Rating a movie is also a tangible indicator of interest in the movie. From a public goods point of view, people might see more benefit to editing movies they are interested in, reducing the net cost of contributing. The collective effort model also predicts that people will be more motivated to perform tasks where they value the outcome. Correcting information for a movie they one cares about is more likely than correct-

ing a random movie, or one that MovieLens *thinks* one will care about but has not yet seen.

The MovieLens experiment suggests that task routing is useful for member-maintained communities. However, editing movie information requires little expertise and modest effort. We were able to work inside of MovieLens and exert a great deal of experimental control. Also, our strategies were very simple. To increase our confidence that intelligent task routing is generally useful, we decided to test more interesting strategies in a less controlled environment that required more effort from contributors. This led us to Wikipedia.

Studying Wikipedia as an encyclopedia

Wikipedia has sparked researchers' curiosity, especially in the areas of knowledge management and computer-mediated communication. As Lih says, "Unlike typical creative efforts, no proof of identity or qualifications is needed to participate and a reputation tracking system is not used within the community. Given the description of how a wiki works, visitors to Wikipedia are often surprised the site works at all." [25] But work it does. Wikipedia exists in dozens of languages, ten of which have over 100,000 articles. Many people contribute: almost 23,000 people made at least five contributions each to the English version in December 2005. Measuring and understanding the growth of Wikipedia is itself an interesting question to some researchers [41].

Wikipedia makes much of its content available for offline analysis through occasional XML dumps of its database. The information is released under several copyright licenses, primarily the GNU Free Documentation License. Wikipedia produces both *current* and *full* dumps. Current dumps are snapshots of the database that contain the most recent version of each page. It is straightforward, though slow, to create a local mirror of Wikipedia by importing a current dump into a copy of MediaWiki, the software that runs Wikipedia. Full dumps contain every revision of every page in Wikipedia, including information about who edited which pages. A full dump of the English version is very large—the May 18, 2006 dump is about 700 GB uncompressed.

This historical data has a number of uses, including computing summary statistics [41] and analyzing patterns of behavior, such as the quick reversion of malicious edits [40]. Several researchers have used this data to understand the quality of Wikipedia articles. Lih used the number of edits, editors, and distinct editors as factors for predicting the quality of articles [25], a direct application of the notion of edit wear [16]. Emigh and Herring analyzed formality of language to compare Wikipedia and Everything2 to other encyclopedias [10]. Stvilia et al. built models using computed features such as median revert time and text readability scores to predict whether an article is high quality [39]. Humans currently tag articles that need work; these models could direct people's energy toward improving articles rather than flagging problems. Similar metrics might be useful for intelligent task routing in other systems as well.

Studying Wikipedia as a community

Wikipedia is not just an encyclopedia; it is also a community. A number of researchers have studied how and why people participate in building the encyclopedia and become part of the community. Ciffolilli and Stvilia et al. compared Wikipedia to the open source movement [4, 39]. In this account, open content and open source are similar, and people participate for the same kinds of reasons: learning, status, belonging, and so on. Lakhani and von Hippel focused on how learning from others reduces the cost of contributing to a forum devoted to support issues around the Apache web server [20]. This finding dovetails nicely with our earlier analysis that reducing contribution costs is important.

Bryant et al. studied how people become regular contributors to Wikipedia [2]. They use activity theory [31] and the notion of legitimate peripheral participation in communities of practice [23] to analyze interviews with nine experienced Wikipedians. Contributors progress from novices who work mostly on editing articles that interest them to experienced users who work both on articles and on meta-level tasks such as mediating disputes, determining policies, and building the Wikipedia community. One important way novices grow toward doing meta-level tasks is by discovering tools useful for those tasks in the course of their normal editing activities. If intelligent task routing can help people find more articles to edit in Wikipedia, they might grow more quickly into experienced contributors. For now, SuggestBot only recommends articles to edit. In principle, it could suggest other tasks such as categorizing articles, resolving editing disputes, and welcoming newcomers.

We extend others' efforts to understand and quantify contribution behavior in Wikipedia by trying to increase contributions through recommendations. Because Wikipedia is a community, we had to act as part of the community, respecting the policies and the norms of Wikipedia. Both imposed constraints on our experimental manipulations. We detail our experience to help other researchers interested in exploring Wikipedia as a research test-bed.

SuggestBot is, as the name suggests, a *bot*. Wikipedia bots are user accounts that computer programs use to make semi-automated or automated edits. Most bots perform mechanical tasks such as expanding template text and repairing links. Bots are governed by a simple policy[1]: "The burden of proof is on the bot-maker to demonstrate that the bot is harmless, is useful, is not a server hog, and has been approved." Bots that do not follow the rules are blocked by administrators. SuggestBot makes some compromises to avoid being a server hog. For instance, it only checks whether an article needs work every few weeks. Ideally it would check that every article it recommends still needs work, but this would require downloading the full text of every recommendation. We also did our development and testing against a local Wikipedia mirror to reduce our impact on the live site.

Bots must be approved by a committee of experienced users, many of whom have written bots themselves. The process consists of posting a description of the bot and reacting to feedback until the committee gives approval. Part of the approval discussion for SuggestBot centered on discovering users' interests. Many users keep *watchlists* of pages they

[1] http://en.wikipedia.org/wiki/WP:B

Welcome!

Hello DonnEdwards, and welcome to Wikipedia! Thank you for your contributions. I hope you like the place and decide to stay. Here are a few good links for newcomers:

- The five pillars of Wikipedia
- How to edit a page
- Help pages
- Tutorial
- How to write a great article
- Manual of Style

I hope you enjoy editing here and being a Wikipedian! Please sign your name on talk pages using four tildes (~~~~); this will automatically produce your name and the date. If you have any questions, check out Wikipedia:Where to ask a question or ask me on my talk page. Again, welcome!

Mushroom 12:13, 18 November 2005 (UTC)

Looking for articles to work on?

Hello, DonnEdwards. I'm SuggestBot, a Wikipedia bot that helps new members contribute to Wikipedia. Based on your previous edits, I have made a list of articles you might like to work on. I hope you find this useful. -- SuggestBot 13:53, 5 January 2006 (UTC)

Figure 2: The pilot version of SuggestBot making suggestions to DonnEdwards. Note the message might be one of many on a user's talk page; here, it is below a welcoming message.

monitor for changes. This sounded promising, but an experienced user pointed out a flaw.

> "I've reviewed the things on my watchlist, and I'm not so sure that they are representative of my interests (at least some things are listed because they were things I thought that should be deleted, some I thought should be merged, and some were things that I can no longer recall why they were originally watched, etc.)"

Misunderstanding Wikipedia norms caused us to make two bad decisions. Our original plan was to use SuggestBot to help people who had contributed a few times become regular contributors. Wikipedia has a *welcoming committee* that gives new editors information on how to make effective contribution and encourages them to do so. Welcome messages and other user-to-user communication are posted to *user talk pages*. These pages are used to resolve editing disputes, organize collaborations, praise, criticize, and just plain be social. Figure 2 shows two messages for Wikipedia user DonnEdwards, one from the welcoming committee and the other from SuggestBot.

One bad decision was to randomly choose who received recommendations in an effort to exert experimental control and avoid self-selection bias. This seemed reasonable because we were following the welcoming committee's lead. New editors don't choose to be welcomed—they simply receive messages from welcoming committee members. But there is a big difference between SuggestBot and the welcoming committee. New users who receive welcomes can find out about the welcoming committee, understand its purpose, discover that it is an established social feature on Wikipedia,

Work type	Description	Count
STUB	Short articles that are missing basic information	355,673
CLEANUP	Articles needing rewriting, formatting, and similar editing	15,370
MERGE	Related articles that may need to be combined	8,355
SOURCE	Articles that need citations to primary sources	7,665
WIKIFY	Articles whose text is not in Wikipedia style	5,954
EXPAND	Articles longer than stubs that still need more information	2,685

Table 1: Work types that SuggestBot recommends, along with an approximate count of articles that need each type of work as of May 2006.

and find that many of its members are longtime contributors. New users who received SuggestBot suggestions could find out that it was an experimental feature run by a member who had very little history in Wikipedia. This surely worked against SuggestBot. Almost everything on Wikipedia is opt-in: users decide whether to edit an article, whether to join a Wikiproject (a group of members who work on a common task), whether to become more involved in the community, etc. We changed SuggestBot from a push to a pull model, giving suggestions only to people who requested them.

Our second bad decision was to post short notes on users' talk pages instead of putting the suggestions directly on the talk page. We feared we would be perceived as spammers and wanted to minimize that perception. This was a mistake. Requiring an extra click to access the actual suggestions most likely reduced the number of people who saw them. Further, the purpose of the talk page is to communicate; adding a layer of indirection was unusual. The current version of SuggestBot posts suggestions directly to a user's talk page.

SuggestBot

We now describe the current version of SuggestBot. Figure 3 gives an overview of its architecture. SuggestBot has four major pieces: pre-processing Wikipedia dumps, modeling users' interests, finding candidate articles to recommend, and making the recommendations. We describe each in turn.

Pre-processing

SuggestBot does extensive pre-processing of historical Wikipedia data. It creates a local mirror of a current dump, then augments it with tables that support recommendations based on text similarity and following links between articles. It also uses the full dump, extracting information about who edits which articles to support a recommendation engine based on co-editing activity. Finally, it does pre-processing to find articles that need work. As mentioned earlier, people tag articles they think need certain kinds of work such as improving writing, expanding content, adding references, and so on. There are dozens of such tags; SuggestBot recommends articles that need six kinds of work, as shown in Table 1. We chose these work types because they are the most frequently used work types on Wikipedia. Another Wikipedia

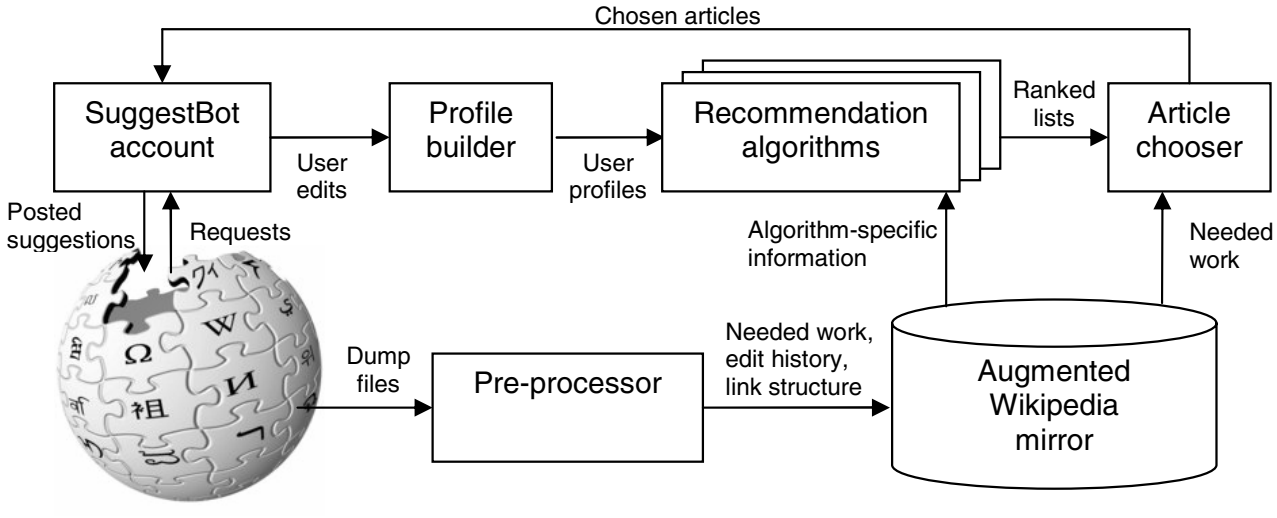

Figure 3: SuggestBot's architecture. It pre-processes Wikipedia data, combining it with user profiles based on edit history as input to several recommendation engines. It combines the results of the engines in order to make suggestions.

bot, Pearle, maintains lists of articles that need each type of work. Rather than re-implement this functionality, SuggestBot uses Pearle to track which articles need work. Stubs are an exception. Pearle does not keep a full list of stubs, so SuggestBot extracts stubs from the full dump file.

Modeling interests

To make recommendations for a user, SuggestBot must model the user's interests. We had to make two key choices: how to represent interest profiles, and whether to construct profiles explicitly or implicitly. We chose to represent a profile as a set of article titles rather than, say, extracting keywords, because several of our recommendation algorithms are most naturally expressed by connections between articles. We also chose implicit over explicit profiles. Explicit profiles are less noisy than implicit ones, but asking users to create interest profiles imposes costs [30]. It is often difficult for people to explicitly express their interests. Web search is a notorious example [38], even for experts [17].

Our next task was to choose behaviors that indicate interest in an article. Wikipedians engage in a variety of behaviors we considered using as implicit interest indicators:

- Reading articles. However, Wikipedia does not publish reading activity. This may be just as well; the comScore World Metrix Internet tracking system estimated that Wikipedia had 131 million unique visitors in May 2006.
- Keeping watchlists. Watchlists would likely be more useful than lists of articles read, but are not publicly available. Also, as described earlier, some people use watchlists to track articles that do not reflect their interests.
- Making contribution lists. Some members explicitly list articles they have created or contributed to on their personal pages. However, most do not, and there is no standard format for listing one's contributions.
- Editing articles. Anyone can view the complete editing history of any Wikipedia user by choosing the "User contributions" link from that user's personal page. This information is also available from dump files.

Because of the problems with using these other behaviors, we chose to use editing articles as our implicit interest indicator. When SuggestBot receives a user's request for recommendations, it fetches from Wikipedia a list of articles that user has edited. It ignores edits to pages that are not articles, such as talk pages. To avoid hogging the server, it retrieves no more than 500 articles per user.

After retrieving the list of articles, it performs simple filtering. As with most implicitly built profiles, there is noise. People often edit articles they have no personal interest in, for example, when they revert vandalism. Vandalism often occurs to the most popular and most controversial articles. For people who perform a lot of vandalism reversion,[2] some recommender algorithms are prone to suggest controversial and often-edited items, as this early user pointed out: "[SuggestBot] is sending me to the most controversial articles on WP. Crazy." We added a filter that removes edits that appear to be vandalism reversion. Experienced users mark vandalism reversion by adding "rv" or "revert" to their comments, so SuggestBot ignores these edits.

Finally, SuggestBot treats multiple edits of an article as a single edit, collapsing the list of remaining edits into a set. Doing so gives each edited article equal weight and improves performance, especially for the text-based and links-based recommenders. A user's profile, then, is the set of article titles left after removing reversions and duplicates.

Finding candidate articles

After building a profile, SuggestBot chooses articles to recommend. In MovieLens, choosing movies a user had rated

[2]Sadly, this is such a common task that Wikipedia has a team called the Counter-Vandalism Unit. See http://en.wikipedia.org/wiki/Wikipedia:Counter-Vandalism_Unit.

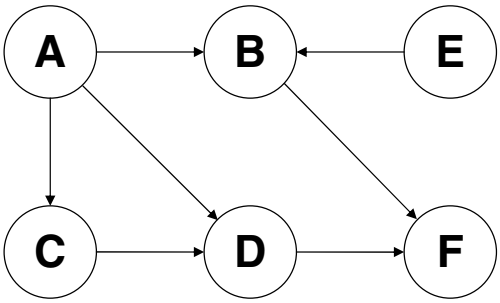

Figure 4: A hypothetical set of links between a set of articles. For a user who has only edited A, after two expansion steps the profile will be $(A = 1, B = 2, C = 2, D = 3, E = 0, F = 2)$.

greatly increased contributions. A naive mapping of this finding to Wikipedia would recommend articles people had already edited. Editors, however, hated seeing this kind of recommendation:

> "...11 of the 34 articles suggested were actually started by me, and I have edited one of the others as well...it might be more useful to exclude articles that one had already edited."

Some MovieLens members reported frustration when they were recommended movies they had already edited. Therefore, we only allow SuggestBot to recommend articles a user has never edited before. But, which articles? We believe recommending items people had rated was effective in MovieLens because this strategy often suggested movies that people were interested in and knew about. SuggestBot assumes people are most likely to be interested in and know about articles that are related to those they have previously edited. SuggestBot's recommendation algorithms implement three different notions of relatedness: similarity of text, explicit connections through links, and implicit connections through co-editing activity. We discuss each in turn.

Text similarity The text similarity recommender creates a keyword query by concatenating the titles of articles in the user's profile and removing stopwords. It uses MySQL 4.1's built-in fulltext search against the full text of articles to retrieve a list of recommended titles. We chose this text search engine mainly for convenience; MediaWiki uses MySQL already and we hope some day to add SuggestBot to MediaWiki. To improve performance, we built a reduced version of the index, eliminating common and very uncommon terms, and used article titles as queries instead of the full text of articles.

Explicit links The links recommender exploits links users have created between articles. This is similar to using citation networks in recommending research papers [8, 28]. Pseudocode for the links recommender is shown in Algorithm 1. It sees Wikipedia as a directed graph where articles are nodes and links are edges. It starts with the set of articles in a user's profile, then expands the profile by following links. It ignores date-related links such as "1970" or "June 15" that lead to large lists of unrelated articles. Articles get credit each time they are found during the expansion process. Articles found early on get more credit because, at every expansion step, all links from articles in the profile are followed, including links that had already been traversed at prior steps.

Algorithm 1 Links recommender.

Param: T {the set of items in target user t's profile}
Param: N {number of requested recommendations: 2500}
Param: $MaxD$ {the maximum depth to crawl: 2}
Param: $BestL$ {preferred number of links to articles: 20}
  {Initialize items in the profile to have a score of 1.}
  for all items i **do**
    $i.score \leftarrow 1$ if $i \in T, 0$ otherwise
  end for
  $depth \leftarrow 0$
  {Expand profile until we have enough articles}
  while $depth < MaxD$ and ($|i|$ with $i.score > 0) < N$ **do**
    for all links to items l from items with $i.score > 0$ **do**
      $l.score \leftarrow l.score + 1$
    end for
    $depth \leftarrow depth + 1$
  end while
  {Remove items from original profile.}
  for all items $i \in T$ **do**
    $i.score \leftarrow 0$
  end for
  {Penalize items with many or few links.}
  for all items i with $i.score > 0$ **do**
    $L \leftarrow number\ of\ links\ to\ i\ in\ Wikipedia$
    $i.score \leftarrow i.score/log(count\ of\ articles/abs(BestL - L))$
  end for
Return: The first N items sorted by descending $i.score$

Figure 4 shows an example where the user has only edited item A. After initializing, the scored profile is $(A = 1, B = 0, C = 0, D = 0, E = 0, F = 0)$. To expand the profile, the algorithm follows all links from every item in the profile. In this case, the expanded profile is $(A = 1, B = 1, C = 1, D = 1, E = 0, F = 0)$. After a second expansion step, the profile is $(A = 1, B = 2, C = 2, D = 3, E = 0, F = 2)$. Note that the link from A to C is followed twice, once at each step. In this example, E will never be recommended because no articles link to it. Future variations of the algorithm might look at in-links as well as out-links.

Profile expansion stops when the needed number of articles have been found. After expansion, the recommender removes articles the user has edited and sorts the remaining articles in descending score order. At first this score was simply how often the article was found during expansion, but pilot testing showed this led to recommending articles with many links. These are poor recommendations because they are usually on popular topics and have been edited by many people. We implemented a penalty function similar to the inter-document frequency term from the TF-IDF formula [36]. However, TF-IDF gives the most credit to terms that occur least often. This led to recommending obscure articles with few links. We modified the function to prefer arti-

cles with a moderate number of links, specified by the $BestL$ parameter in Algorithm 1. We asked several users to evaluate lists of recommendations at different levels of $BestL$, settling on 20 based on their reactions.

Algorithm 2 Co-edit recommender.

Param: $MinJ$ {minimum user similarity: 0.0001}
Param: $MinC$ {minimum similar editors per item: 3}
Param: N {number of requested recommendations: 2500}
Param: T {the set of items in target user t's profile}
  for all items i **do**
    $i.score \leftarrow 0, i.count \leftarrow 0$
  end for
  {Find all my neighbors}
  for all users u who have edited any item $i \in T$ **do**
    $U \leftarrow$ all items edited by u
    $J \leftarrow \frac{|T \cap U|}{|T \cup U|}$ {Jaccard similarity with this neighbor}
    {only recommend if similar enough}
    if $J > MinJ$ **then**
      for all items $i \in U$ **do**
        $i.score \leftarrow i.score + J$ {weighted credit}
        $i.count \leftarrow i.count + 1$
      end for
    end if
  end for
  {Remove items edited by few others, or edited by t}
  for all items i with $i.score > 0$ **do**
    if $i.count < MinC$ or $i \in T$ **then**
      $i.score \leftarrow 0$
    end if
  end for
Return: The first N items sorted by descending $i.score$

Co-editing patterns The co-edit recommender uses collaborative filtering to find people whose edit history is similar to that of a target user. SuggestBot uses a variation of the classic user-user algorithm from [35]. Instead of explicit 1-to-5 ratings of articles, the co-edit recommender treats editing an item as an implicit positive rating. This is sometimes called a unary rating scale, and is often found in e-commerce applications [15]. Common similarity metrics such as Pearson correlation and cosine similarity [37] are not well-suited to unary data because they only consider items both users have rated. With a unary scale, this rating will always be identical, and all users who have at least one item in common will be perfectly similar. Algorithms often penalize similarity based on few common ratings by *significance weighting* [14]; for example, by multiplying the computed similarity by $n/50$ if the two users have $n < 50$ ratings in common. Instead of this ad-hoc penalty, we use the Jaccard metric for set similarity between profiles, defined as $J(A, B) = |A \cap B|/|A \cup B|$. This gives credit for having items in common while penalizing large profiles with low overlap.

The full algorithm for the co-edit recommender is shown in Algorithm 2. We find all users who have rated something in common with target user t and compute t's similarity to each of those users. Then, for each other user u, we give credit to every item u has edited based on how similar u is to t. At the end, we return the N items with the highest scores; these are the items most likely to be edited by other people who are similar to t. The algorithm ignores users who are not very similar to t, and throws away items that are edited by few other people. We chose low thresholds of 0.0001 and 3, respectively, because many users have edited few items and their pool of potential neighbors is small. As with the links recommender, the co-edit recommender tends to recommend popular and controversial items that many people edit. Rather than search for an appropriate recommender-specific penalty, we imposed a policy for all engines: do not recommend any item in the top 1 percent of most-edited articles.

Note that in all cases, we used straightforward variations of our algorithms. The coedit recommender, for instance, might take advantage of collaborative filtering extensions such as incorporating taxonomies [42], combining multiple recommenders with hybrid algorithms [3, 28], and exploring trust and reputation metrics between users [33]. Though these extensions often improve performance, our hope was that straightforward implementations would produce significant results. Such implementations are quicker to code and test, easier for other designers to adopt, and simpler to eventually integrate with the MediaWiki software.

Making recommendations

Once the engines produce their lists of recommendations, the article chooser combines and filters them. The chooser has 34 slots for articles to recommend per request: 19 stubs and three each of the other five work types. These numbers make for a clean interface that fills most of a web browser window, as shown in Figure 1. For each slot, the chooser randomly picks an engine, then asks it for its highest-ranked recommendation that:

- needs the required type of work for the slot,
- is not in the top 1% of most frequently edited articles, and
- has not already been chosen for another slot.

If the engine cannot make a recommendation, the chooser randomly tries another engine. If all engines fail, it chooses a random article title that needs the required type of work. This happens about one percent of the time, because to improve performance the article chooser asks each engine for only 2,500 recommendations.

Deploying SuggestBot: results

As of September 2006, about 50 people per week request articles to edit, even though we do not advertise. People discover SuggestBot by seeing suggestions on other users' talk pages, by seeing it in the recent changes list, and by back-channel communication between users. The first sizeable batch of users came from a post an early user made to a Wikipedia mailing list. In March 2006, another user added a link to SuggestBot on the Community Portal. In six months, over 1,200 people have received suggestions, and SuggestBot has received dozens of positive comments.

But is intelligent task routing useful? Do users actually edit suggested articles? Even if they do, does being intelligent matter? Perhaps people are as likely to edit randomly chosen articles as personalized ones, and it is the act of asking that matters. If task routing is more useful than random, which algorithms are most promising? Can we easily remove noise

Recommender	Edited	Total	Percent
Co-edit	29	726	4.0%
Text	34	790	4.3%
Links	25	742	3.4%
Random	8	836	1.0%
Total	96	3,094	3.1%

Table 2: Suggestions edited within two weeks of posting. Personalizing recommendations improves performance compared to random ($\chi^2(2, 3094) = 16.77, p < 0.01$).

Recommender	Edited	Total	Percent
Co-edit	33	1,152	2.9%
Text	36	1,195	3.0%
Links	29	1,140	2.5%
Total	98	3,487	2.8%

Table 3: Suggestions edited within two weeks of posting for the three intelligent algorithms. There are no significant differences between the algorithms ($\chi^2(2, 3487) = 0.49, p = 0.78$).

from profiles? To address these questions, we observed the editing behavior of SuggestBot users in three experiments.

Our first experiment compared the intelligent algorithms to random suggestions. We gave the random recommender the same chance to be chosen as the other recommenders. Using random recommendations as a baseline may sound like a straw man—but Wikipedia does randomly choose articles to feature on the Community Portal. Every subject saw some recommendations from each engine. Overall, 91 users received a total of 3,094 suggestions. Table 2 shows how many of each recommender's suggestions were edited at least once by the recipient within two weeks of the suggestions being posted. The intelligent algorithms did about four times as well as random. A chi-square test shows this difference is significant.

The random recommender was so bad that we now use it only as a backup if the intelligent algorithms all fail. Our second experiment tried to detect differences between the intelligent algorithms. Again, we used a within-subjects design and observed how many suggestions are edited within two weeks. A total of 103 subjects saw 3,487 non-random suggestions. Table 3 shows the results. The links-based recommender appears to do slightly worse than the text and co-edit recommenders, but there were no significant differences.

Although the overall performance of the individual recommenders was similar, they failed under different conditions. The text recommender often failed by focusing on one relatively rare word in an article title and returning a series of recommendations containing that word. The links recommender was sometimes tricked by category templates that contain links to the other pages in the category, as in Figure 5. Since every page in such categories points to every other page in the category, these pages would appear many times as the links recommender expanded the profile. The

Figure 5: Articles in categories where every page links to every other page in the category sometimes confused the links recommender, causing it to recommend the other pages in the category.

Filtered?	Edited	Total	Percent
Yes	91	2,448	3.7%
No	103	2,720	3.8%

Table 4: Suggestions edited within two weeks of posting, with minor edits filtered from some profiles. There were no significant differences between filtered and unfiltered profiles ($\chi^2(2, 5168) = 0.02, p < 1$).

co-edit recommender had a tendency to recommend often-edited articles. Since these failure modes are independent, using meta-search techniques to combine their results might improve SuggestBot's performance.

Our third experiment dealt with trying to improve user profiles by removing noise, a general problem for implicitly constructed profiles. Wikipedians often make small changes to articles, and can flag such edits as *minor*. Not everyone uses the minor flag consistently, but many users asked if SuggestBot could ignore minor edits. Their intuition was that eliminating minor edits from profiles would improve their recommendations. We tested this intuition by removing minor edits for some users but not others. A total of 152 subjects saw 5,168 suggestions. Table 4 shows that removing minor edits had no effect on contributions.

A methodological note

As a general rule, we value deploying designs via field experiments to explore whether they work in real contexts. However, researchers must be careful when comparing designs deployed sequentially. For example, people edited fewer suggestions in the second experiment than in the first. It is tempting to conclude that the lower number of edits may be due to the self-selection bias we feared; perhaps the most interested and committed Wikipedia users signed up for the first experiment. However, the percentage of edited suggestions increased from the second experiment to the third. That suggests that self-selection bias did not cause the drop in the second experiment.

The problem is that context can change rapidly, making it hard to know whether differences in behavior are caused by

differences in the design. A number of conditions changed between the second and third experiments. Wikipedia grew by over 20% during that time. SuggestBot also changed, using newer Wikipedia data and an improved algorithm for filtering out revert edits. Conditions changed between the first and second experiments as well—for example, someone posted the link to SuggestBot on the community portal page during this time period. Thus, we do not try to compare results between experiments or combine their results.

Instead, researchers using field experiments should deploy multiple designs at once whenever possible. This is what we did in each of our three experiments. In experiments 1 and 2, we used a within-subjects design, while in experiment 3 we used a between-subjects design. In all three cases, we deployed the designs simultaneously. This maximized the chance that our results were based on differences in the designs rather than changes in the context.

Conclusion

Our results in both prior research with MovieLens and in the present research with Wikipedia show that intelligent task routing can dramatically increase members' contributions. The two systems are different in a number of ways, including size, the nature of contributions, domain, and sociability. In both cases, simple algorithms gave strong results, increasing contributions by about four times. We saw these strong results in field experiments where we built real designs for real communities with real users and real problems to solve. Deploying our designs in Wikipedia imposed some constraints on what we could do, but provided compensating benefits in the form of copious offline data to analyze and insight from experienced members of the community. The designs themselves are rooted in established social science theories of motivation that are based on laboratory experiments.

Based on this foundation, we are confident that many other communities stand to enjoy similar benefits from applying intelligent task routing. For example: Slashdot assigns comments randomly to meta-moderators. Why not choose comments that are similar to one's own posts? Or, why not ask people to meta-moderate comments from stories they have read? Likewise, many posts in discussion forums go unanswered. Research has shown that people are more likely to return to post again if they receive a response [18]. Communities that want to encourage people to return could route unanswered posts to experienced members whose past contributions are most similar to the unanswered question.

Intelligent task routing is likely to be most useful for communities where tasks are numerous and heterogeneous. Like information filtering, task routing will grow more valuable as the number of tasks increases. It is also likely to be more valuable as the diversity of tasks increases—although even in MovieLens, where the only task is editing movie information, simple algorithms produced large benefits.

Task routing is promising, but there is room for improvement. The absolute numbers of articles edited in Wikipedia are not as high as we would like. Tuning our algorithms, bringing more sophisticated algorithms to bear, and using ideas from meta-search to combine the recommenders' output are logical next steps. Learning to remove noise from profiles is a general problem. Our lack of results with obvious strategies suggests it is an interesting problem. Solutions could benefit both SuggestBot and recommender systems in general. Finally, we might improve SuggestBot by developing interfaces that give users more control over their profiles with minimal perceived cost, as McNee et al. did [29] for new users of recommender systems.

Still, SuggestBot is a useful tool, and intelligent task routing is a useful strategy for eliciting contributions from members of online communities. Designers should use intelligent task routing in order to build more valuable communities and better experiences for the people who inhabit them.

REFERENCES

1. G. Beenen, K. Ling, X. Wang, K. Chang, D. Frankowski, P. Resnick, and R. E. Kraut. Using social psychology to motivate contributions to online communities. In *Proc. CSCW2004*, Chicago, IL, 2004.
2. S. L. Bryant, A. Forte, and A. Bruckman. Becoming wikipedian: transformation of participation in a collaborative online encyclopedia. In *GROUP '05*, pages 1–10, 2005.
3. R. Burke. Hybrid recommender systems: Survey and experiments. *UMUAI*, 12(4):331–370, 2002.
4. A. Ciffolilli. Phantom authority, self-selective recruitment and retention of members in virtual communities: The case of wikipedia. *First Monday*, 8(12), 2003.
5. T. Connolly and B. Thorn. Discretionary databases: Theory, data, and implications. *Organizations and Communication Technology*, pages 219–233, 1990.
6. D. Cosley, D. Frankowski, S. Kiesler, L. Terveen, and J. Riedl. How oversight improves member-maintained communities. In *CHI '05*, Portland, OR, 2005.
7. D. Cosley, D. Frankowski, L. Terveen, and J. Riedl. Using intelligent task routing and contribution review to help communities build artifacts of lasting value. In *CHI'06*, Montreal, CA, 2006.
8. D. Cosley, S. Lawrence, and D. M. Pennock. REFEREE: An open framework for practical testing of recommender systems using researchindex. In *VLDB'02*, Hong Kong, August 20–23 2002.
9. D. Cosley, P. Ludford, and L. Terveen. Studying the effect of similarity in online task-focused interactions. In *GROUP'03*, pages 321–329, 2003.
10. W. G. Emigh and S. C. Herring. Collaborative authoring on the web: A genre analysis of online encyclopedias. In *HICSS*, 2005.
11. R. Hardin. *Collective Action*. Johns Hopkins, Baltimore, 1982.
12. S. G. Harkins. Social loafing and social facilitation. *Journal of Experimental Social Psych.*, 23:1–18, 1987.

13. F. M. Harper, X. Li, Y. Chen, and J. A. Konstan. An economic model of user rating in an online recommender system. In *User Modeling*, page 307, Edinburgh, Scotland, 2005. Springer Berlin.

14. J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR '99*, 1999.

15. J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM TOIS*, 22(1):5–53, Jan. 2004.

16. W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless. Edit wear and read wear. In *Proceedings of CHI 1992*, pages 3–9, 1992.

17. C. Höscher and G. Strube. Web search behavior of internet experts and newbies. In *WWW '00*, pages 337–346, 2000.

18. E. Joyce and R. E. Kraut. Predicting continued participation in newsgroups. *JCMC*, 11(3), 2006.

19. N. L. Kerr. Motivation losses in small groups: a social dilemma analysis. *J. Pers. Soc. Psych.*, 45:819–828, 1983.

20. K. Lakhani and E. von Hippel. How open source software works: "free" user-to-user assistance. *Research Policy*, 32:923–943, 2002.

21. C. Lampe and P. Resnick. Slash(dot) and burn: distributed moderation in a large online conversation space. In *CHI '04*, pages 543–550, 2004.

22. B. Latané, K. Williams, and S. Harkins. Many hands make light the work: The causes and consequences of social loafing. *J. Pers. Soc. Psych.*, 37:822–832, 1979.

23. J. Lave and E. Wenger. *Situated learning: Legitimate peripheral participation*. Cambridge University, New York, NY, 1991.

24. J. O. Ledyard. *Public Goods: A Survey of Experimental Research*, pages 111–194. The Handbook of Experimental Research. Princeton University Press, Princeton, NJ, 1995.

25. A. Lih. Wikipedia as participatory journalism: Reliable sources? metrics for evaluating collaborative media as a news resource. In *5th International Symposium on Online Journalism*, Austin, TX, 2004.

26. K. Ling, G. Beenen, P. Ludford, X. Wang, K. Chang, X. Li, D. Cosley, D. Frankowski, and L. Terveen. Using social psychology to motivate contributions to online communities. *JCMC*, 10(4), 2005.

27. P. J. Ludford, D. Cosley, D. Frankowski, and L. Terveen. Think different: increasing online community participation using uniqueness and group dissimilarity. In *CHI '04*, pages 631–638, 2004.

28. S. McNee, I. Albert, D. Cosley, P. Gopalkrishnan, S. K. Lam, A. M. Rashid, J. Konstan, and J. Riedl. On the recommending of citations for research papers. In *CSCW '02*, November 2002.

29. S. M. McNee, S. K. Lam, J. A. Konstan, and J. Riedl. Interfaces for eliciting new user preferences in recommender systems. In *User Modeling*, pages 178–187, Johnstown, PA, USA, 2003. Springer Verlag.

30. M. Morita and Y. Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *SIGIR '94*, page 48, 1994.

31. B. A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, Cambridge, Mass., 1996.

32. G. B. Newby and C. Franks. Distributed proofreading. In *Proc. JCDL2003*, pages 361–363, 2003.

33. J. O'Donovan and B. Smyth. Is trust robust?: An analysis of trust-based recommendation. In *IUI '06*, pages 101–108, New York, NY, USA, 2006.

34. A. M. Rashid, K. Ling, R. D. Tassone, P. Resnick, R. Kraut, and J. Riedl. Motivating participation by displaying the value of contribution. In *Extended Abstracts of CHI '06*, 2006.

35. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *CSCW '94*, pages 175–186, 1994.

36. G. Salton and M. McGill. *Introduction to Modern Information Retrival*. McGraw-Hill, 1983.

37. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW '01*, pages 285–295, 2001.

38. C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

39. B. Stvilia, M. Twidale, L. Smith, and L. Gasser. Assessing information quality of a community-based encyclopedia. In *Proc. International Conference on Information Quality*, pages 442–454, 2005.

40. F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *CHI '04*, pages 575–582, 2004.

41. J. Voss. Measuring wikipedia. In *Proc. International Conference of the International Society for Scientometrics and Informetrics*, Stockholm, 2005.

42. C.-N. Ziegler, G. Lausen, and L. Schmidt-Thieme. Taxonomy-driven computation of product recommendations. In *CIKM '04*, pages 406–415, 2004.