Proceedings of the Sixth International AAAI Conference on Weblogs and Social Media

Automatic Versus Human Navigation in Information Networks

Robert West and **Jure Leskovec**
Computer Science Department
Stanford University
{west, jure}@cs.stanford.edu

Abstract

People regularly face tasks that can be understood as navigation in information networks, where the goal is to find a path between two given nodes. In many such situations, the navigator only gets local access to the node currently under inspection and its immediate neighbors. This lack of global information about the network notwithstanding, humans tend to be good at finding short paths, despite the fact that real-world networks are typically very large. One potential reason for this could be that humans possess vast amounts of background knowledge about the world, which they leverage to make good guesses about possible solutions. In this paper we ask the question: Are human-like high-level reasoning skills really necessary for finding short paths? To answer this question, we design a number of navigation agents without such skills, which use only simple numerical features. We evaluate the agents on the task of navigating Wikipedia, a domain for which we also possess large-scale human navigation data. We observe that the agents find shorter paths than humans on average and therefore conclude that, perhaps surprisingly, no sophisticated background knowledge or high-level reasoning is required for navigating the complex Wikipedia network.

Introduction

Mankind has been creating, maintaining, and using information networks for a long time. Examples from the offline world include scientific papers citing each other and dictionaries with cross-referenced entries. With the emergence of the online age, we have been witnessing a migration of essentially all things written onto the Web, which may itself be considered a gigantic information network that contains, in turn, a myriad of smaller, more structured subnetworks. Navigating these resources is a task people are faced with on a daily basis: we surf Wikipedia to find the answers to questions; we browse the virtual shelves of online stores by clicking from one article description to the next; or we explore social networking sites to find the name of that lovely person at yesterday's party.

This type of navigation can be phrased as search in a graph, where we gradually move across edges from a start to a target node. However, identifying short paths by means of, say, Dijkstra's algorithm is impossible, since this would require knowledge of the full graph, whereas typically we have only local access to the network, seeing only the current and the previously visited nodes, as well as their direct neighbors. Other than that, neither the—typically large—set of all existing nodes (often with complex content) nor the links between them are known ahead of time. Therefore, it is often easy to get lost. Nevertheless, people are rather good at finding short paths between pairs of nodes, a fact known as the 'small world' phenomenon (Milgram 1967). One potential explanation is that people are good at 'connecting the dots' because they have common sense and a large body of background knowledge for reasoning about the world, such that their search can be guided by intuitive expectations about the unknown link structure of the network. For instance, one might think that, in order to find a path from the Wikipedia article about MOZART to that about TERMINATOR, one must know that Mozart is from the same country as the person enacting the Terminator, such that trying to find a path like ⟨MOZART, AUSTRIA, SCHWARZENEGGER, TERMINATOR⟩ would make sense.

But how hard is navigating content-rich information networks really? Does the task require high-level reasoning skills and common sense—which computers do not have yet, and which is a very ambitious goal? Or can it be formalized in a straightforward way such that even a simple computer program can solve it? In the present paper we attempt to shed light on this question by designing a number of automatic agents without human-like knowledge and reasoning skills, which use only simple numerical features. We evaluate these agents in the controlled example task of Wikipedia navigation. In this setting, the goal is to find a short path from a given start to a given target article by clicking only existing links. We choose Wikipedia as the validation domain for two reasons. First, it contains rich knowledge of the kind people are good at reasoning about. Second, we previously designed an online human-computation game, called *Wikispeedia,* through which we have been collecting over 30,000 examples of humans navigating the Wikipedia network and where the player's goal is also to find short link chains between given start and target articles.

When designing the search algorithm and the features available to the agents, we have striven to allow for as fair a comparison between humans and automatic agents as possible. Analyzing the results, we observe that even simple algorithms can find shorter paths on average than humans.

While humans need twice the optimal number of clicks on average, our best automatic agent achieves a factor as low as 1.5. Also, the frequency with which human searches are over twice as long as shortest paths is 4 times as high as for automatic searches. Hence we conclude that, maybe surprisingly, very basic numerical features suffice for navigating a complex network such as Wikipedia.

On a more faceted level, human and automatic search are qualitatively different. Our evaluation lends support to the conclusion that, while the agents are more efficient on average, humans are less likely to get totally lost during search. Humans typically form robust high-level plans with backup options, something our automatic agents cannot do (for a detailed analysis of how humans solve this task, cf. West and Leskovec 2012). Instead, the agents compensate for this lack of smartness with increased thoroughness: since they cannot know what to expect, they always have to inspect all options available, thereby missing out on fewer immediate opportunities than humans, who, focused on executing a premeditated plan, may overlook shortcuts.

While we consider these findings interesting in their own right, our algorithms may also prove useful in the context of social and information networks from a practical perspective. For instance, 'social browsing' (Lerman and Jones 2007) is an emerging paradigm. Also, it is estimated that 99.8% of the Web's content, the so-called 'deep Web', is hidden 'behind the query forms of searchable databases' (He et al. 2007) and therefore cannot be systematically indexed by general search engines, making it necessary to navigate by clicking from page to page. Even on the 'surface Web', there is little use in today's search engines whenever we cannot formulate a concise query describing what we are looking for (think of that lovely person at yesterday's party); and even if a decent query was entered, the results often only serve as the starting point for further, click-based navigation. Furthermore, sometimes the entire path that connects two pages rather than a single result page matters; e.g., the answer to a question might be spread over a series of Wikipedia articles. Finally, some information resources, such as peer-to-peer networks, are deliberately designed in a decentralized manner, such that global search is impossible. In all these scenarios, navigation-based search algorithms may complement the use of query-based search engines. By analyzing the learned importance weights attributed by our algorithms to different features, we can quantify what information matters most for solving search tasks, which may in turn help us design tools for the support of human browsing.

We proceed as follows. First, we introduce and motivate the abstract search algorithm, before describing concrete implementations of it, which differ only in the way they score the neighbors for choosing the next click. Thereafter, we compare our algorithms among each other and to humans, and analyze the relative importance of features. Then we review related work, and finally conclude by discussing our findings and their implications.

The abstract search algorithm

The task we investigate is how to navigate from a *start* to a *target* node by traversing edges of a given graph. Since our goal is to design algorithms whose performance we can compare to that of humans in a fair manner, it is important that we subject the automatic agents to the same restrictions a human navigator faces.

First, it is a requirement that in deciding where to go next, our agents use only local node features independent of global information about the network. Only the nodes visited so far, their immediate neighbors, and the target may play a role in picking the next node. Second, typical Web-browsing software also imposes restrictions. The user can only follow links from the current page or return to the immediate predecessor by clicking the back button. Jumping between two unconnected pages is not possible, even if they were both visited separately before.

These constraints give rise to a simple navigation algorithm. As input, it takes the start node s, the target node t, and an evaluation function V. The latter is used to score each neighbor u' of the current node u and, due to the locality constraint, is a function of u, u', and t only. We write the value of u's neighbor u' as $V(u'|u,t)$. Ideally, V should capture the likelihood of u' being closer to the target than u is. At each step, the algorithm chooses from those neighbors of the current node u that have not previously been visited, picking the candidate u' with the highest value $V(u'|u,t)$. If all neighbors of u were previously visited, or if it has none, we backtrack, which is the analogue of a human's clicking the back button. In the worst case, the algorithm has to explore all nodes reachable from s in order to find t. Pseudocode for the basic navigation procedure is listed in Algorithm 1.

Algorithm 1 may be considered an abstract meta-algorithm, since the order in which the neighbors of the current node are visited is determined by an arbitrary evaluation function V. The concrete choice of V is, of course, crucial for the performance of the algorithm. We experiment with several choices of V, which we describe next.

Algorithm 1 Basic navigation algorithm

Input: start node s; target node t; evaluation function V
Output: sequence of nodes visited

```
push s onto initially empty stack S
while stack S is not empty do
    pop node u from stack S and remember it as visited
    if u = t then
        return sequence of nodes visited
    else if u has not been visited before then
        for all neighbors u' of u, in order of increasing value according to V(u'|u,t) do
            push u' onto stack S
            if u' = t then
                break // make sure t is picked in next iteration
            end if
        end for
    end if
end while
return 'no path between s and t exists'
```

Implementations of the search algorithm

For the evaluation function V, we experiment with simple heuristics, as well as a set of machine learning methods.

Heuristic agents

The most basic agents navigate according to a single, simple criterion. In particular, degree-based (Adamic et al. 2001) and similarity-based (Kleinberg 2000) methods have been proposed.

Degree-based navigation (DBN). For the degree-based agent, we define $V(u'|u,t) := \deg(u')$ (the outdegree of u'). Note that neither the current node u nor the target t are considered. This agent is based on the intuition that nodes with many neighbors are a good choice because they are more likely to contain a direct link to t, or at least offer many ways of continuing the search.

Similarity-based navigation (SBN). For the similarity-based agent, we let $V(u'|u,t) := \text{tf-idf}(u',t)$, which is defined as the standard TF-IDF cosine of the textual contents of the candidate u' and the target t. The choice of similarity as a feature is based on the notion of *homophily,* a property shared by many real-world networks: the more similar two nodes are, the more likely they are to link to each other.

Expected-value navigation (EVN). Şimşek and Jensen (2005) combine degree and similarity in a simple probabilistic model, giving rise to 'expected-value navigation'. They make the assumption that each endpoint w of a node v's outlinks is sampled independently of the others with probability q_{vw}, which is a function of the similarity of v and w. The value of a candidate u' is then defined as the probability that at least one outlink of u' ends in the target t, i.e.,

$$V(u'|u,t) := 1 - (1 - q_{u't})^{\deg(u')}. \quad (1)$$

Following Şimşek and Jensen, we assign all pairs of nodes (v,w) to discrete bins based on tf-idf(v,w), and estimate q_{vw} as the fraction of pairs in the respective bin that are direct neighbors.

Machine learning agents

Next, we introduce machine learning agents that all combine features linearly in a weighted sum but differ in how the weights are learned. Thus, they are more adaptable to the navigation domain at hand than the hard-coded heuristics.

The agents learn a separate weight vector (and thereby a separate evaluation function V_i) for each path position i; i.e., the i-th iteration of the while loop of Algorithm 1 uses V_i to evaluate neighbors.

Supervised learning. The first class of learning agents we discuss is based on supervised learning. In this scenario, the evaluation function may be seen as a classifier measuring the confidence that a given click is a 'good' choice. Training is done for each path position independently and is straightforward: sample pairs (u,t); for each pair, choose a number of 'good' and an equal number of 'bad' neighbors of u; fit the weights. The way in which we define 'good' and 'bad' determines the ultimate behavior of the agent, and we experiment with two options: In the first approach, u's neighbor u' is defined as 'good' if it decreases the shortest-path length to t (we call this 'lucrative vs. non-lucrative').[1] Our second approach is applicable when human search paths are available. Then we may define u' as 'good' if the link from u to u' was ever clicked by a human under target t, and as 'bad' otherwise. This will make the agent behave similarly to human players ('human vs. non-human').

For weight-fitting, we experiment with two methods. First, *logistic regression* (LR) is a standard choice for binary classification. Second, a slightly more sophisticated classifier arises from the observation that, at each step, Algorithm 1 ever only picks the neighbor with the highest value. So choosing a neighbor may be considered a ranking problem, where we require that the top-ranked candidate be good. In other words, whereas LR maximizes accuracy, we in fact want to maximize precision at k with $k = 1$. A host of learning methods are available for learning to rank. In particular, we experiment with SVM-MAP (Yue et al. 2007), a *ranking support vector machine* that chooses the weights that maximize the mean average precision (MAP) of the induced ranking.

Reinforcement learning (RL). Supervised techniques require labeled ground-truth data. Either we need global network information (the shortest-path lengths between all node pairs) or large amounts of human click behavior data. Also, these algorithms work offline: first, all the training data are processed for learning weights; then, the algorithm is run with fixed weights.

A more natural paradigm for teaching an agent how to behave in a given environment can be found in reinforcement learning (RL) (Sutton and Barto 1998). Here, the shortest-path lengths of node pairs need not be known ahead of time. The agent starts navigating with random weights, and only once it happens to find the target does it learn through positive feedback that the recent choices were good and adapts its weights to make similar choices more likely in the future. Therefore, the RL agent works online—it acts and learns at the same time.

In an RL setting, an agent moves from state to state by performing actions, at each step receiving a numerical reward (or punishment) indicating whether the current state is desirable (or not). In network navigation, states are nodes and actions are link clicks. The reward $r(u)$ received for reaching state u is positive if $u = t$ and negative otherwise. Consider a path $\langle s = u_1, ..., u_n = t \rangle$ produced by running Algorithm 1. The cumulative reward obtained from u_i onward is defined as $R(u_i) := \sum_{k \geq i} r(u_k)$. That is, the shorter the path, the higher the cumulative reward (since all summation factors but the last are negative), and reaching the target t from u_i as quickly as possible is equivalent to maximizing the cumulative reward $R(u_i)$. Let $R^*(\cdot)$ be the $R(\cdot)$ achieved by an optimal agent. Then, if we manage, for all k, to find weights $\mathbf{w}_k$ such that $V_k(u_k|u_{k-1},t) = \mathbf{w}_k^\top \mathbf{f}(u_k|u_{k-1},t)$ is a good approximation of $R^*(u_k)$, then choosing the neighbor u_{i+1} of

[1] Recall that the agents are allowed only local access to the network at navigation time, whereas shortest-path length, needed to assemble this type of training set, is a global notion. However, this information is needed only at training, not at navigation time, and in many scenarios this might be legitimate, since the agent might get full access to a small training portion of the entire network.

the current u_i that maximizes $V_{i+1}(u_{i+1}|u_i,t)$ will tend to result in short solution paths (where $\mathbf{f}(u_{i+1}|u_i,t)$ is the feature vector of u_{i+1}, given the current node u_i and the target t).

To learn the weights, a temporal difference (TD) update is performed in each iteration of the while loop, after pushing all neighbors of the current node onto the stack (i.e., between lines 12 and 13 of Algorithm 1):

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha\left[r(u_i) + V_{i+1}(u_{i+1}|u_i,t) - V_i(u_i|u_{i-1},t)\right] \cdot \\ \cdot \mathbf{f}(u_i|u_{i-1},t), \quad (2)$$

where α is a learning rate parameter, u_i the current node, and u_{i+1} the node on top of the stack, i.e., the one that will be visited next. This update is also used in the so-called SARSA algorithm (Rummery and Niranjan 1994). Instead of deriving it, we explain it in intuitive terms. Recall that we strive to achieve $V_k(u_k|u_{k-1},t) \approx R^*(u_k)$, for all k. In equation (2), $V_i(u_i|u_{i-1},t)$ is our estimate of $R^*(u_i)$ before receiving the reward $r(u_i)$, and $r(u_i) + V_{i+1}(u_{i+1}|u_i,t)$ is our estimate for it afterwards, so the update adjusts the weights in a gradient-descent way, making $V_i(u_i|u_{i-1},t)$ match the true $R^*(u_i)$ more closely.

To learn the weights, we have the agent solve a number of training navigation tasks, where start and target nodes are picked randomly. In the first task, the agent starts with random weights. Subsequently, task $j+1$ starts with the weights resulting from task j. During testing, weights are fixed and not updated any more.

Empirical evaluation

The above descriptions of our agents are abstract in the sense that we have not specified which features we consider. The appropriateness of features obviously depends on the specific network to be navigated. We now describe our evaluation network, list and analyze the features used, and compare the agents among each other as well as to humans.

Experimental setup

The Wikipedia network. We evaluate our navigational agents on the Wikipedia hyperlink network, under the task of finding short paths between given pairs of Wikipedia articles. Wikipedia is an attractive validation domain mainly for two reasons. First, there is large-scale human navigational data for the Wikipedia network, which we have been collecting through the online game *Wikispeedia*[2] (West, Pineau, and Precup 2009), where the task formulation is the same as for our agents: find as short a path as possible from a given s to a given t, only by clicking links (or the back button). Over 30,000 trajectories have been gathered from around 9,400 distinct IP addresses, such that it is possible to compare the automatic agents to humans quantitatively. Second, Wikipedia contains a wealth of rich knowledge of the kind that humans can easily deal with: they may use all their world knowledge and reasoning skills to find clever paths between the start and target articles; recall the example task 'MOZART to TERMINATOR' from the introduction. Since our agents do not possess such sophisticated, structured knowledge, the comparison sheds light on the question whether navigating rich knowledge resources is possible without human-like knowledge.

Our online game uses the 2007 Wikipedia Selection for schools,[3] which is edited by SOS Children's Villages UK and contains 4,604 articles (and about 120,000 links) that can serve as a free alternative to costly encyclopedias. We sample the start–target pairs, the so-called *missions,* only from the strongly connected core component comprising 4,051 articles (98%), such that every mission has a solution. The mean shortest-path length between article pairs is 3.18, with a median of 3, and a maximum of 9.

Features. In our experiments, we consider the following features (alongside a constant bias term):

1. $\deg(u')$: degree of the candidate u', as in DBN;
2. $\deg(u)$: degree of the current node u;
3. tf-idf(u',t): the TF-IDF cosine of u' and t, as in SBN, is important because of homophily;
4. tf-idf(u,t): the similarity between u and t;
5. tf-idf(u,u'): if this feature gets a large weight, then the next node is preferred to be similar to the current one; otherwise, clicks between unrelated nodes are preferred; e.g., a small similarity between neighbors might be good in the beginning of paths, for quickly getting away from s, a large one towards the end, for homing in on t;
6. linkcos$(\cdot,\cdot)$: the equivalent of features 3–5, but with TF-IDF cosine replaced by *link cosine,* a similarity measure based on the outgoing links of the nodes under consideration: represent a node as the sparse vector of its outgoing links, where the non-zero entries are IDF-weighted; similarity is then defined as the cosine of such vectors (Milne and Witten 2008); the purpose of including this in addition to TF-IDF is to have redundancy: when one fails, the other might still work;
7. taxdist(u',t): the Wikipedia version we use contains a tree-like hierarchy of categories, the leaves of which are populated by the articles; the 'taxonomic distance' between u' and t is defined as the number of edges in the tree that separate the two.

Features 1–6 are considered in logarithmic form, since the distributions of their values are approximately log-normal.

We emphasize that the content of the neighbor u' is not available to the agents explicitly; e.g., while the content of u' is necessary to compute tf-idf(u',t) (feature 3), the agents know only this resulting number, not the content of u' itself. We argue that this way the agents do not have an unfair advantage over humans, who also only get to see the textual content of an article once they visit it. Also, the agents know the degree of the neighbor u' (feature 1), while humans do not. However, humans can probably roughly anticipate the degrees of articles based on intuitions about the importance of the corresponding concepts (more important concepts typically have higher degree), and we think that this difference in available information does not crucially affect the fairness of our comparison either.

[2] http://www.wikispeedia.net (accessed 03/2012)

[3] http://schools-wikipedia.org (accessed 08/2008)

Training set creation for supervised learning. Recall from the section about supervised learning that we consider classifiers for distinguishing 'lucrative vs. non-lucrative' and 'human vs. non-human' clicks, respectively. Whereas for 'human vs. non-human' the examples are necessarily created from human paths, this need not be the case for 'lucrative vs. non-lucrative'. As an alternative we explore using the transitions made on random shortest paths as positive examples, with negative examples subsampled randomly.

Evaluation methodology. As the basic notion of performance, we consider the number of nodes visited until the target is found, which we call *game length* or *search time* (we emphasize that search time is not measured in seconds). It is important to also count nodes from which the agent or human backtracked, since otherwise it would be trivial to always find an optimal solution simply by running breadth-first search.

We count a path as 'long' if its length is more than twice that of an optimal solution. This gives us an idea how often agents and humans do reasonably well.

We report the following quantities:

1. The *percentage of long paths,* which counts how often the agent took over twice the optimal number of steps.
2. The *overhead factor* of games, defined as ℓ/ℓ^* for a game of actual length ℓ and optimal length ℓ^*.

In our online game, humans often give up on missions; at each step, there is a probability of approximately 10% of this happening, such that only 46% of all games begun are finished. This might happen, e.g., because the player gets frustrated, so oftentimes paths that would be long are not even recorded, resulting in an overestimation of performance. To be fair and give the automatic agents the chance to drop out as well, we introduce a small probability ρ of giving up at each step. The missions given up are discarded, which lets us compute less noisy averages that are not skewed by the essentially failed searches in which nearly the entire graph must be explored before finding t (only a small fraction of searches, cf. Fig. 1).

Test sets. We run our agents on two test sets. For the *random test set,* we sampled 100,000 (s,t)-pairs uniformly at random from the strongly connected core component. This large test set allows for tight performance estimates. We choose $\rho = .005$, which implies that a very long search is given up after an expected 200 steps. The probability of giving up within the first 1,000 clicks is 99%, that of not giving up within the first 10 clicks is 95%, such that we exclude the essentially failed searches without giving up too many searches too early.

The *human test set* contains 1,200 missions that were also played by humans, such that we can compare the agents' to human performance. Over 30,000 human paths have been collected through our online game, but we consider only the missions played at least 4 times (median 5 times), such that the estimate of human performance can be made more reliable by taking the median human search time for each mission. Since the agents find short paths for virtually all human missions, no drop-out option is required, and we set $\rho = 0$.

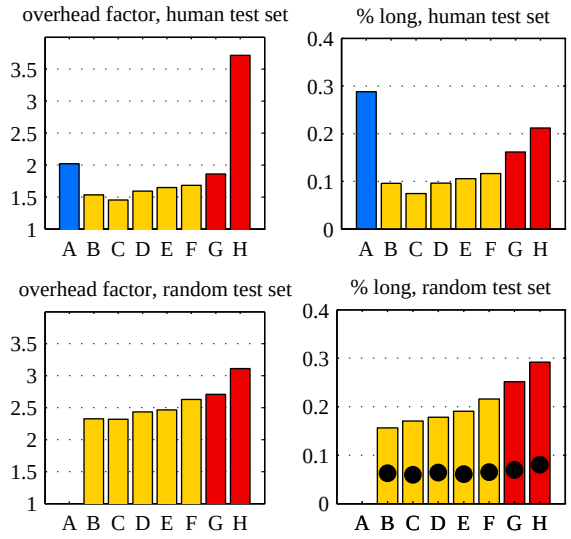

Figure 1: Results of the agent evaluation, with comparison to humans. **Left:** mean overhead factor (ratio of actual to optimal path length); **right:** percentage of long paths (i.e., of more than twice the optimal length); **top:** 1,200 missions played by at least 4 humans each; **bottom:** 100,000 random missions; **blue (A):** humans; **yellow (B–F):** learning agents; **red (G–H):** heuristic agents; **black dots:** percentage of searches given up; **(A)** humans, **(B)** RL, **(C)** SVM-MAP, **(D)** LR (lucrative vs. non-lucrative, human paths), **(E)** LR (lucrative vs. non-lucrative, opt. paths), **(F)** LR (human vs. non-human), **(G)** SBN, **(H)** EVN. Not shown: DBN (overhead factors: 20 [top row] and 51 [bottom row]; percentages long: 0.81 and 0.86; percentage given up: 0.47).

Results

Fig. 1 summarizes the performance of humans (blue), the learning (yellow), and the heuristic agents (red), in terms of the mean overhead factor and the percentage of searches longer than twice the optimum (i.e., lower means better). Error bars are small and therefore not shown.

Mean overhead factor and percentage long. First, the learned agents (yellow) perform better than the heuristic ones (red). In particular, RL (B) and SVM-MAP (C) are best. What is more surprising is that, with the exception of the poorly performing DBN algorithm, even the simple agents do fairly well. In fact, most agents achieve lower mean overhead factors than humans: while humans (blue) take on average twice the optimal time, our best agents need only about 1.5 times the optimum on the same test set (upper left). Whereas 29% of human paths require more than twice the optimal number of steps, this is the case for only about 7% of the searches performed by the best agent (upper right). We also note that missions accomplished by humans appear to be easier: all agents perform better on these missions (top row) than on random ones (bottom row). The reason is probably that humans choose not to play the really hard missions in the first place, or start but do not finish them.

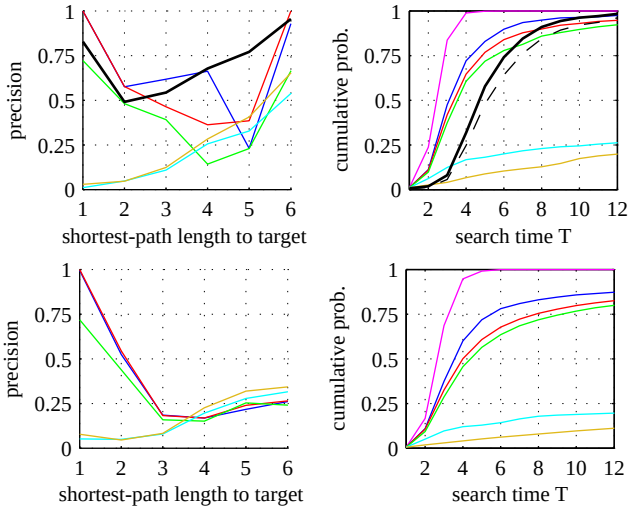

Figure 2: **Left:** Precision (probability of top-ranked neighbor being closer to target than current node); **right:** cdf of search time; **top:** 1,200 missions played by at least 4 humans each; **bottom:** 100,000 random missions; **magenta:** optimal; **blue:** RL; **red:** SBN; **green:** EVN; **cyan:** DBN; **ocher:** random; **black:** human (dashed: hypothetical case of humans never giving up). (Color order corresponds to top-down order in lower right plot.)

In Fig. 2 we take a closer look at how humans differ from agents. We plot only RL (blue) as a proxy for all learning agents, since they show rather similar behavior.

Precision. The left column of Fig. 2 shows the precision of humans and of the agents' evaluation functions V, as a function of shortest-path length (SPL) to t, where precision is defined as the probability that the top-ranking neighbor decreases the SPL to t. Again, the top row shows the human, the bottom row the random test set (the estimates for the random test set are much less noisy because it is 100 times the size). The basic shape of curves is identical for most agents: precision is higher far away from and close to the target, and lower in between. Again, the only exception is DBN (cyan), whose mean precision is as bad as that of a randomly acting agent (ocher); since many targets are not reachable directly from high-degree nodes, DBN is not able to find these targets quickly. The learned agent's precision (blue) exceeds that of humans (i.e., the agent makes good choices more reliably) when the SPL to t is 1, 2, or 3. We note that 69% of all connected pairs have a SPL of at most 3, so one may say this agent makes better picks than humans most of the time.

Cumulative search time distribution. The right column of Fig. 2 shows the cumulative distribution function (cdf) of search time, i.e., the fraction of searches of length at most T. One curve lying entirely above the others may be seen as the respective agent being better all around. The top curve (magenta) constitutes the upper bound of a hypothetical agent that always finds the shortest path; the bottom curve (ocher) is the lower bound set by a randomly acting agent. In between, the ranking is led by the learning agents (blue), SBN (red), EVN (green), and finally, far off, DBN (cyan). We see two reasons for the unexpected fact that plain SBN does

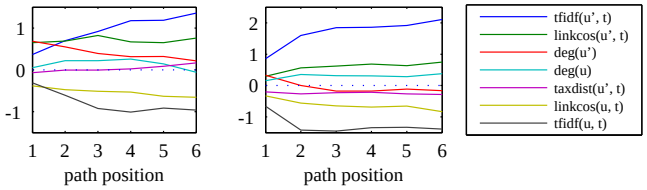

Figure 3: Weights learned by logistic regression; one set of weights per path position. **Left:** classifier 'lucrative vs. non-lucrative'; **right:** 'human vs. non-human' (cf. section about supervised learning); tf-idf(u,u') and linkcos(u,u') are neglected because their weights are close to zero everywhere. (Colors are ordered top-down at $x = 3$ in left plot.)

better than EVN. On the one hand, even the TF-IDF cosine alone is a powerful feature because of Wikipedia's rich textual content and its homophily-based link structure. On the other hand, EVN includes degree information in a fixed way, which does not work well in the case of Wikipedia, possibly because the probabilistic model it adopts (cf. (1)) is not true: link endpoints are not picked independently, e.g., no two links can have the same endpoints in our graph representation of Wikipedia (although a page may contain several anchors linking to the same neighbor, in the graph these are represented as a single edge). Nonetheless, the learners' increased performance proves that taking degree into account can help if done in a more flexible way.

Comparison to humans. Now consider the cdf of search time on the human test set (top right). Virtually all test missions are optimally solvable with at most 4 clicks (magenta). The learning agent manages to solve 75% of them with 4 clicks or fewer, while humans achieve only 30%. However, the agents sometimes fail entirely; when this happens, nearly all of the network has to be searched before t is found. This is why the 100% mark is approached only slowly in the cdf curve. Humans, on the contrary, incur fewer complete failures, manifest in the fact that the human cdf (solid black) approaches the 100% mark faster. One reason for this is, of course, that humans give up rather than visit thousands of nodes. To account for this, we compute an ideal curve for the hypothetical case that humans are forced to finish every mission, based on measured drop-out rates (Dodds, Muhamad, and Watts 2003). We find that even then, the human curve (dashed black) starts to overtake the others around $T = 9$. A potential explanation of this effect could be that humans, while on average further away from optimal than the automatic agents, are less prone to get totally lost.

Feature analysis

In the previous section we found that combining several features linearly with learned weights yields better performance than the heuristic baselines. An analysis of the resulting weights can tell us about the relative importance of the respective features and lets us draw conclusions about what information should be taken into account by real search applications based on our algorithms.

Fig. 3 plots the weights learned by the logistic regression agent as functions of path position (feature values were normalized to mean 0 and variance 1, such that the weights

are comparable). We show both types of classifiers (cf. section about supervised learning): 'lucrative vs. non-lucrative' (left) and 'human vs. non-human' (right). The weights are similar in both cases. Features capturing the similarity between u' and t (blue TF-IDF cosine, green link cosine) dominate on nearly all positions, and even more so as paths progress. This is in tune with the good performance of the SBN agent based on TF-IDF alone.

The weight of the candidate's degree $\deg(u')$ (red) decreases in importance along paths. In the 'lucrative vs. non-lucrative' classifier, it has the strongest weight first but is then overtaken by similarity; in the 'human vs. non-human' classifier, it starts positive but becomes negative after two clicks. While navigating through high-degree hubs is very common in the beginning of paths, people seem to actively avoid it later on. Once searchers are close to t, they often reach it by navigating through specific articles similar to t, which often have low degree. This causes the purely degree-based agent (DBN) to perform so badly and contributes to the lesser performance of EVN: while high-degree nodes are not necessarily desirable, both heuristics always prefer them.

We remark that the basic structure 'degree to get away from s, similarity to home in on t' is in tune with a recent analysis of human Wikipedia navigation (West and Leskovec 2012).

Note that, once the weights have been learned, features not depending on the candidate u' play no role. However, including them during learning may still affect the weights, since the quality of u' may be caused by factors independent of u' itself, and only by including these factors can this be accounted for; e.g., $\text{tf-idf}(u,t)$ and $\text{linkcos}(u,t)$ get large negative weights, as *any* u' linked from a node u with small similarity to t is itself likely to be closer to t than u is.

Related work

We identify three broad areas of related work: decentralized search, focused crawling, and click-trail analysis.

Decentralized search is similar to the setup we study, in that a target node is to be reached via local navigation only. The analysis of such processes was kick-started by Milgram's (1967) seminal 'small world' experiment, later repeated on a larger scale by Dodds, Muhamad, and Watts (2003). Theoretical analyses of the properties a network must have to warrant efficient decentralized search (Kleinberg 2000; Liben-Nowell et al. 2005) showed that just the right amount of homophily is both necessary and sufficient, which gives rise to similarity-based navigation. Degree-based navigation was analyzed by Adamic et al. (2001) and combined with similarity by Şimşek and Jensen (2005). In this paper we experiment with all three heuristics but, more important, also explore the use of learning to trade off degree and similarity optimally rather than heuristically. We also note that our scenario differs from decentralized search in a crucial point: while there, each node represents an autonomous agent who forwards the message once and then loses control, one single participant navigates from start to end in our setup, such that high-level long-term plans can be developed and executed.

Another area related to our work is focused crawling (Chakrabarti, Van den Berg, and Dom 1999), where the goal is to find as many pages as possible within topical boundaries. This differs from our task formulation, where one single page is to be found. Hence, focused crawling algorithms are more akin to breadth-first search than to the greedy navigation scheme our task requires. Nonetheless, the same node features we find to be of central importance in our task—relevance (i.e., similarity to the target) and 'hubness' (i.e., degree)—matter most in focused crawling, too.

Finally, a fairly large body of work deals with the analysis of the click trails of humans navigating the Web and the development of assistive browsing tools. An early such tool was 'WebWatcher' (Joachims, Freitag, and Mitchell 1997); it highlights hyperlinks based on user goals inferred from an initial keyword query and subsequent clicks. The notion of information scent (Chi et al. 2001) has been popular in modeling human navigation behavior; e.g., 'ScentTrails' (Olston and Chi 2003) is a framework that blends query- and navigation-based search smoothly, inspired by the intuition that navigating becomes useful whenever keyword queries are infeasible or unsuccessful. The development of navigation-based search algorithms is further justified by Teevan et al. (2004), who demonstrate that users prefer clicking to querying even when the exact search target is known. Also, White and Huang (2010) show that, even once a query has been issued, further navigation starting on the search-engine result page is attractive because it adds value to the content of the search results themselves. What distinguishes the present work from previous research on click-trail analysis is that those tools are tested in small-scale experiments—e.g., 8 users (Chi et al. 2001)—, and that they focus on the collaboration of humans and computers. On the contrary, we conduct a large-scale comparison of the two, each acting on their own. This allows for a more controlled analysis, the results of which may in turn help to improve future collaborative tools; e.g., the above frameworks are essentially based on content similarity, while our analysis shows that additional features such as node degree are also useful.

Discussion

This paper investigates the question whether structured knowledge and high-level reasoning are necessary for navigating a rich information network such as Wikipedia—a legitimate question, given that connections are oftentimes latent, i.e., not manifest in plain content words. Recall the introductory example mission from MOZART to TERMINATOR with the solution path ⟨MOZART, AUSTRIA, SCHWARZENEGGER, TERMINATOR⟩. The word overlap of the articles on AUSTRIA and TERMINATOR is small, so the plain TF-IDF cosine might not fire in this case. While this problem can probably in many cases be mitigated via generalization techniques such as Latent Semantic Analysis (Landauer and Dumais 1997), this is not even necessary: as shown, even simple agents with bare-bones word-count knowledge and no generalization, let alone reasoning skills, can outperform humans.

Due to the verbose nature of Wikipedia articles, cases as above, in which a semantic connection is not realized ver-

bally, might be rare. Also, even if the agent misses the AUSTRIA opportunity, any other neighbor of MOZART that increases the similarity to TERMINATOR is likely to be a good choice, too. This follows from the homophily present in Wikipedia: previous work (West and Leskovec 2012) showed that Wikipedia links are distributed in a way that theoretically allows similarity-based navigation to find short paths.

Degree information can further help to increase performance, but care must be taken how it is incorporated. When a search begins, it is often beneficial to visit a hub with many links to diverse parts of the graph, but since many targets are reachable only via topic-specific articles of low degree, navigating according to degree only is bound to fail. Our feature analysis shows that degree should be weighted less strongly (and similarity more strongly) on later path positions.

What adds to making the automatic agents better than humans is that the latter often miss good opportunities. Even when the target is only one hop away, they pick the respective link with a probability of only 83% (cf. precision in Fig. 2). A possible explanation of this effect would be that humans typically have commonsense expectations about what links may exist and, based thereupon, form high-level plans regarding the route to take before even making the first click. Following through on their premeditated plans, subjects might often just skim pages for links they already expect to exist, thereby not taking notice of shortcuts hidden in the abundant textual article contents. For instance, in the above example, there might for some reason be a link from AUSTRIA directly to TERMINATOR, but as the searcher has set her mind on the more reasonable chain through SCHWARZENEGGER, she may not even become aware of that link's existence. We also hypothesize that humans' deeper understanding of the world is the reason why their searches fail completely (i.e., take an extremely long time) less often: instead of exact, narrow plans, they sketch out rough, high-level strategies with backup options that are robust to contingencies. Analyzing subjects' strategy-making behavior to decide to what extent these hypotheses are true remains an interesting avenue of future research.

Our evaluation is restricted to Wikipedia, since one of the principal goals of this paper is to compare automatic to human navigation and since, to the best of our knowledge, Wikipedia is the only network for which large-scale goal-directed human browsing data is available. (Vast numbers of click trails are also collected, e.g., by browser toolbars, but most such trails are not goal-directed, and even for those that are, the target is not explicitly known.) We expect that our algorithms are equally applicable to other information networks with rich content, e.g., social, citation, or peer-to-peer networks, or even the 'deep Web'. While further experimentation is required to confirm this intuition, we anticipate that our algorithms could be used to design search interfaces that combine the strengths of humans and computers in a symbiosis of intuition and precision.

We deem our reinforcement learning approach particularly attractive, as it reasons directly about actions. Thus, if it is to be used to navigate more general networks than Wikipedia, it is possible to include additional actions other than link clicks, e.g., submitting a form, posting a question to a Q&A site, requesting help through a crowdsourcing service, or querying the user for feedback. Extending our algorithm this way is certainly not trivial but might be a promising step towards creating smarter programs with the ability of exploring and acting on the Web more autonomously.

Acknowledgements. This research was in part supported by NSF CNS-1010921, IIS-1016909, IIS-1149837, Albert Yu & Mary Bechmann Foundation, Boeing, IBM, Lightspeed, Samsung, Yahoo, Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship.

References

Adamic, L. A.; Lukose, R. M.; Puniyani, A. R.; and Huberman, B. A. 2001. Search in power-law networks. *Phys. Rev. E* 64(4).

Chakrabarti, S.; Van den Berg, M.; and Dom, B. 1999. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks* 31(11-16).

Chi, E. H.; Pirolli, P.; Chen, K.; and Pitkow, J. 2001. Using information scent to model user information needs and actions and the Web. In *CHI-01*.

Dodds, P. S.; Muhamad, R.; and Watts, D. J. 2003. An experimental study of search in global social networks. *Science* 301(5634).

He, B.; Patel, M.; Zhang, Z.; and Chang, K. C.-C. 2007. Accessing the deep Web. *CACM* 50(5).

Joachims, T.; Freitag, D.; and Mitchell, T. 1997. WebWatcher: A tour guide for the World Wide Web. In *IJCAI-97*.

Kleinberg, J. 2000. Navigation in a small world. *Nature* 406(6798).

Landauer, T., and Dumais, S. T. 1997. A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psych. Rev.* 104(2).

Lerman, K., and Jones, L. A. 2007. Social browsing on Flickr. In *ICWSM-07*.

Liben-Nowell, D.; Novak, J.; Kumar, R.; Raghavan, P.; and Tomkins, A. 2005. Geographic routing in social networks. *PNAS* 102(33).

Milgram, S. 1967. The small world problem. *Psychology Today* 2.

Milne, D., and Witten, I. H. 2008. An effective, low-cost measure of semantic relatedness obtained from Wikipedia links. In *WIKIAI-08*.

Olston, C., and Chi, E. H. 2003. ScentTrails: Integrating browsing and searching on the Web. *TCHI* 10(3).

Rummery, G. A., and Niranjan, M. 1994. On-line Q-learning using connectionist systems. Technical report, University of Cambridge.

Şimşek, Ö., and Jensen, D. 2005. Decentralized search in networks using homophily and degree disparity. In *IJCAI-05*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge University Press.

Teevan, J.; Alvarado, C.; Ackerman, M. S.; and Karger, D. R. 2004. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *CHI-04*.

West, R., and Leskovec, J. 2012. Human wayfinding in information networks. In *WWW-12*.

West, R.; Pineau, J.; and Precup, D. 2009. Wikispeedia: An online game for inferring semantic distances between concepts. In *IJCAI-09*.

White, R. W., and Huang, J. 2010. Assessing the scenic route: Measuring the value of search trails in Web logs. In *SIGIR-07*.

Yue, Y.; Finley, T.; Radlinski, F.; and Joachims, T. 2007. A support vector method for optimizing average precision. In *SIGIR-07*.