ComPlexUs
NETWORK MODELLING
Complexus 2006;3:158–168 DOI: 10.1159/000094197
Published online: August 25, 2006

Is Selection Optimal for Scale-Free Small Worlds?

Zs. Palotai[a] Cs. Farkas[b] A. Lőrincz[a]

[a]Department of Information Systems, Eötvös Loránd University, Budapest, Hungary;
[b]Department of Computer Science and Engineering, University of South Carolina, Columbia, S.C., USA

Key Words
Scale-free small world · No free lunch theorem · Internet

Abstract
The 'no free lunch theorem' claims that for the set of all problems no algorithm performs better than random search and, thus, selection can be advantageous only on a limited set of problems. In this paper we investigate how the topological structure of the environment influences algorithmic efficiency. We study the performance of algorithms, using selective learning, reinforcement learning, and their combinations, in random, scale-free, and scale-free small world (SFSW) environments. The learning problem is to search for novel, not-yet-found information. We ran our experiments on a large news site and on its downloaded portion. Controlled experiments were performed on this downloaded portion: we modified the topology, but preserved the publication time of the news. Our empirical results show that the selective learning is the most efficient in SFSW topology. In non-small world topologies, however, the combination of the selective and reinforcement learning algorithms performs the best.

Fax +41 61 306 12 34
E-Mail karger@karger.ch
www.karger.com
KARGER

1424–8492/06/0033–0158
$23.50/0
Accessible online at:
www.karger.com/cpu

A. Lőrincz
Department of Information Systems, Eötvös Loránd University
Pázmány Péter sétány 1/c, HU–1117 Budapest (Hungary)
Tel. +36 1 209 0555/8473, Fax +36 1 381 2140,
E-Mail andras.lorincz@elte.hu

Simplexus

A free lunch after all?

Developers of web search engines and data mining tools expend vast sums attempting to find the most efficient ways for users to search. Mighty algorithms travail indexes with great speed and seemingly great efficacy, narrowing a key word search to a 'short' list of results within seconds. However, the 'no free lunch theorem' holds that there simply is no algorithm that can perform better than a random selection on the set of all problems. In other words, no matter how hard those developers try they will simply never beat a randomly selected set of 'results'.

In the present paper, Palotai, Farkas, and Lőrincz seek to understand how the topological structure of the environment influences algorithmic efficiency and whether or not there might be a 'free' lunch, after all. They compare the performance of algorithms in unearthing news from a large news website, using basic learning techniques – selective learning, reinforcement learning (RL), and their combinations, in random, scale-free, and scale-free small world (SFSW) environments. Their empirical results suggest that selective learning is the most efficient in SFSW topology, but in non-small world topologies, combining selective and RL algorithms gave the best results.

Evolving systems, both natural and artificial (like the Web), exhibit scale-free or SFSW properties. Previous researchers have shown that there is no performance difference between optimization or search algorithms if the algorithms are tested on every possible problem. This implies that differences in the performance of specific algorithms are simply a result of specific properties of the problem being looked at. By uncovering these properties, it should then be possible to develop optimized search approaches, despite the no free lunch theorem. The structure of a database or index is an important property and others have already demonstrated that an evo-

1 Introduction

According to the 'no free lunch theorem' (NFL theorem) [1] there is no performance difference between optimization or search algorithms if we test the algorithms on every possible problem. Therefore, it seems that improved performances of specific algorithms, such as selection, are consequences of specific problem properties. Finding these properties will aid the development of optimized solutions. Recent research shows that evolving structures, both natural and artificial (like the Web), exhibit scale-free or scale-free small world (SFSW) properties [2, 3]. Annunziato et al. [4] show that a particular evolutionary algorithm has a better performance in artificial scale-free environment than in lattice, small world, or random environments. That is, the structure of the environment has an impact on the efficiency of the algorithms.

We have designed a controllable experiment and compared the performances of different algorithms on different structures. We aimed to keep the complexity of experiments and the number of designer-specifiable parameters minimal. We considered the structure of the environment, the algorithms, and the fitness values. Our choice for the structure was the World Wide Web (WWW). The WWW is considered the largest source of rapidly changing data. It has an SFSW structure [2, 5].

The everyday usage of the Web is the search for novel information. Therefore, we have a natural reward function; the number of novel documents that the algorithms can find. We consider this property as one of the most important components of our work: *we did not specify the temporal and structural details of the reward system.*

We ran controlled experiments on a time stamped and downloaded portion of a large WWW news site. This allowed us to preserve the temporal structure of the rewards, but also supported the modification of the underlying connectivity structure, i.e., how the novel information can be found.

Our agents were Web crawlers or foragers. Crawlers travel from link to link foraging new, not-yet-seen information. Our agents used the simplest possible algorithms. Selective learning concerned the selection and memorization of good links. We also used a simple version of reinforcement learning (RL) [for a review of RL, see, e.g. 6]. RL was used alone, and was also combined with selective learning. Our choice of RL is motivated by its structural and algorithmic simplicity and that RL is concerned with the optimization of the expected value of long-term cumulated profit (LTP) [for a recent review on evolutionary computing, see 7; for reviews on related evolutionary theories and the dynamics of self-modifying systems, see 8, 9 and 10, 11, respectively]. Similar concepts have been studied in other evolutionary systems where organisms compete for space and resources and cooperate through direct interaction [see, e.g. 12 and references therein].

Hybrid algorithms have a long history. A well-known example is the TD-Gammon program of Tesauro [13]. The author applied MLP function approximators for value estimation in RL. RL has also been used in concurrent learning problems like ours: robots had to learn to forage together via direct interaction [14]. Another combination of the algorithms concerns evolutionary learning embedded into the framework of RL to improve decision making [15–18].

It is important to note that communication and competition among our foragers are indirect. Only the first submitter of a document may receive positive reinforcement and this is the only interaction among the crawlers. Our work is different from other studies using combinations of genetic, evolutionary, function approximation, and RL algorithms, in that (1) it does not require explicit fitness function; (2) we do not have control over the original environment; (3) we can change the environment in a reproducible fashion; (4) collaborating lutionary algorithm that adapts to the results it obtains performs better in artificial scale-free environments than in lattice, small world, or random environments. Knowing this could be exploited usefully in developing more effective data mining and web search tools.

Everyday millions of users carry out millions of searches on the Web, looking, on the whole, for novel information. The search itself therefore bears its own natural reward function, explains Lőrincz, the number of novel web pages or documents that the algorithms can find. The fact that the researchers did not, and could not, specify the temporal and structural details of this reward system is perhaps the most important aspect of the current paper. With this in mind the team set about designing an experiment to compare algorithms' approaches. Their controllable experiment was designed so that the complexity of experiments and the number of designer-specifiable parameters were minimal and looked at the structure of the search environment, the algorithms, and the fitness values. They chose the web as the most appropriate data set as it is a large source of rapidly changing data and because of its organic evolution has an almost entirely SFSW structure. They chose a large news website containing time-stamped entries and downloaded a portion of this with which to experiment. They could thus preserve the temporal structure of the rewards, and at the same time modify the underlying connectivity structure so that the way in which novel information can be found could be changed to test different search methods. They used a group of computer agents – Web crawlers. These travel from link to link and forage for new, not-yet-seen information. The agents used the simplest possible algorithms: selective learning concerned the selection and memorization of good links. They also used a simple version of RL. RL was used alone and in conjunction with selective learning in separate experiments.

individuals use value estimation under 'evolutionary pressure', and (5) individuals work without direct interaction with each other. The crawler system is a self-assembling system, which is made of adaptive components, and the communication between components is kept to as little as possible.

The paper is organized as follows. We review the related web crawler tools, including those [19–21] that our work is based upon, in section 2. We describe our algorithms and the forager architecture in section 3. This section contains the necessary algorithmic details imposed by the task, the search of the Web. We present our experiments on the Web and the controlled simulations in section 4. Discussions can be found in section 5. Conclusions are drawn in section 6.

2 Related Work

There are important problems that have been studied in the context of crawlers. Angkawattanawit and Rungsawang [22], and Menczer [23] study topic-specific crawlers. Risvik and Michelsen [24] address research issues related to the exponential growth of the Web. Cho and Garcia-Molina [25], Menczer [23] and Edwards et al. [26] study the problem of different refresh rates of URLs (possibly as high as hourly or as low as yearly).

An introduction to and a broad overview of topic-specific crawlers are provided [22]. They propose to learn starting URLs, topic key words and URL ordering through consecutive crawling attempts. They show that the learning of starting URLs and the use of consecutive crawling attempts can increase the efficiency of the crawlers. The used heuristic is similar to the weblog algorithm [21], which also finds good starting URLs and periodically restarts the crawling from the newly learned ones. The main limitation of this work is that it is incapable of addressing the freshness (i.e., modification) of already visited Web pages.

Menczer [23] describes some disadvantages of current Web search engines on the dynamic Web, e.g., the low ratio of fresh or relevant documents. He proposes to complement the search engines with intelligent crawlers or web-mining agents to overcome those disadvantages. He introduces the InfoSpider architecture that uses genetic algorithm and RL, and also describes the MySpider implementation of it, which starts from the 100 top pages of AltaVista. Our weblog algorithm uses local selection for finding good starting URLs for searches, thus does not depend on any search engines. Dependence on a search engine can be a suffer limitation of most existing search agents, like MySpiders [23]. Note, however, that it is an easy matter to combine the present algorithm with URLs offered by search engines.

Risvik and Michelsen [24] overview different dimensions of web dynamics and show the arising problems in a search engine model. The main part of the paper focuses on the problems that crawlers need to overcome on the dynamic Web. As a possible solution the authors propose a heterogeneous crawling architecture. The main limitation of their crawling architecture is that they must divide the web to be crawled into distinct portions manually before the crawling starts. A weblog-like distributed algorithm – as suggested here – may be used in that architecture to overcome this limitation.

Cho and Garcia-Molina [25] define mathematically the freshness and age of documents of search engines. They propose the Poisson process as a model for page refreshment. The authors also propose various refresh policies and study their effectiveness both theoretically and on real data. They present the optimal refresh policies for their freshness and age metrics under the Poisson page refresh model. The authors show that these policies are superior to others on real data, too. Although they show that in their database more than 20% of the documents are

There are two types of agents – foragers and reinforcing agents (RA). Foragers crawl the web and send back the addresses (URLs, uniform resource locators) of the selected documents to the RA. The RA is a simple machine that sends out foragers one after the other an equal number of times. It acts as a central agent that can determine which forager finds new information first and then reinforces that forager. The RA sends reinforcements to the foragers based on the received URLs. Lőrincz and his colleagues have employed a fleet of foragers to study the competition among individual foragers. The foragers then compete with each other to find the most relevant documents and so efficiently collect new relevant documents without interacting directly with each other. Foragers adapt based on reinforcements using their learning algorithms. The central agent has been used to model real life, where there is no actual agent, but where food that is eaten in any given area by one forager is obviously then unavailable to foragers that follow, so they are not reinforced by searching the same region.

The researchers then applied a set of constraints on finding the minimal set of algorithms. As such, the algorithms should allow the identification of unimportant parameters, support the specialization of individual foragers, allow the melding of evolutionary learning and individual learning, and minimize communication as much as possible. The team deployed sixteen foragers to search the CNN website for relevant documents, that is pages that are less than 24 h old and that have not been previously marked as seen. In this the download part of the experiment, the goal was to assimilate a database that is unbiased from the point of view of later modifications to its structure. The first eight forages used both selection-based and RL. The other eight foragers used only selection-based learning. The selection-based algorithm, known as the weblog update algorithm, learns the best starting URLs of

changed each day, they disclosed these documents from their studies. Their crawler visited the documents once each day for 5 months, and thus cannot measure the exact change rate of those documents. In our work, however, we definitely concentrate on these frequently changing documents.

3 Forager Architecture

There are two different kinds of agents: the foragers and the reinforcing agent (RA). The fleet of foragers crawls the web and sends the URLs of the selected documents to the RA. The RA determines which forager should work for the RA and how long a forager should work. The RA sends reinforcements to the foragers based on the received URLs.

We employ a fleet of foragers to study the competition among individual foragers. A forager has simple and limited capabilities, like a stack for a limited number of starting URLs and a simple, content-based URL ordering. The foragers compete with each other to find the most relevant documents. In this way they efficiently and quickly collect new relevant documents without direct interaction.

At first we present the basic algorithms, followed by the algorithms for the RA and the foragers.

3.1 Algorithms

Our constraints on finding the minimal set of algorithms were as follows: The algorithms should (1) allow the identification of unimportant parameters, (2) support the specialization of the individuals (the foragers), (3) allow the joining of evolutionary learning and individual learning, and (4) minimize communication as much as possible. We shall return to these points in section 5.

Weblog Algorithm and Starting URL Selection

A forager periodically restarts from a URL randomly selected from the list of starting URLs. The sequence of visited URLs between two restarts forms a path. The starting URL list is formed from the 10 first URLs of the weblog. In the weblog there are 100 URLs with their associated weblog values in descending order. The weblog value of a URL estimates the expected sum of rewards during a path after visiting that URL. The weblog update algorithm modifies the weblog before a new path is started. The weblog value of a URL already in the weblog is modified toward the sum of rewards (*sumR*) in the remaining part of the path after that URL:

$$new\ Value = (1 - \beta)\ old\ Value + \beta\ sumR,$$

where β was set to 0.3. A new URL has the value of the actual sum of rewards in the remaining part of the path. If a URL has a high weblog value it means that around that URL there are many relevant documents. Therefore, it may worth it to start a search from that URL.

Without the weblog update algorithm the weblog and thus the starting URL list remains the same throughout the searches. The weblog algorithm is a very simple version of evolutionary algorithms. Here, evolution may occur at two different levels: the list of URLs of the forager is evolving by the reordering of the weblog. Also, a forager may multiply, and its weblog, or part of it, may spread through inheritance. This way, the weblog algorithm incorporates the basic features of evolutionary algorithms. This simple form shall be satisfactory for our purposes.

RL-Based URL Ordering

A forager can modify its URL ordering based on the received reinforcements of the sent URLs. The (immediate) profit is the difference of received rewards and penalties at any given step. Immediate profit is a myopic characterization of a step to a URL. Foragers have an adaptive continuous value estimator and follow the policy that maximizes the expected LTP instead of the immediate profit. Such estimators searches from which the forager can periodically restart its search. The RL-based URL ordering update algorithm modifies the crawling directions of the foragers, i.e., the type of food (here, topic) they like. Once foraging was complete the team investigated the link structure of returned documents and found it to obey a power-law function, in other words it was scale free.

The next step was to repeat the experiment but in a simulation mode rather than on the web. For this, the forager architecture was implemented in Matlab as if it were running on the same computer used in the 'real' experiments. They conducted simulations with four different kinds of foragers in each environment: WR foragers used both the weblog update and the RL-based URL ordering update algorithms. WL foragers used only the weblog update algorithm without URL ordering update; these foragers each had a different variable weight factor. RL foragers used only the RL-based URL ordering update algorithm without the weblog update algorithm. Finally, fix foragers did not use the weblog update and the RL-based URL ordering update algorithms. These foragers had fixed starting URLs and fixed weights.

The researchers found that the efficiency of the algorithms depends strongly on the weight vectors associated with each forager in the simulation. In contrast, the number of starting points has little impact on efficiency. Two foragers with twenty different starting points were comparable in efficiency to sixteen foragers using 160 different starting points. The number of starting points and the number of foragers are less important than the weight vectors in the simulations. Larger numbers of crawlers and having more than one computer can only change the situation weakly, which gives the researchers confidence that their studies of structural dependencies would be valid.

They found that freshness and age values of different foragers change in the

can be easily realized in neural systems [6, 27, 28]. Policy and profit estimation are interlinked concepts: profit estimation determines the policy, whereas policy influences choices and, in turn, the expected LTP [for a review, see 6]. Here, choices are based on the greedy LTP policy. The forager visits the URL, which belongs to the frontier (the list of linked but not yet visited URLs, see later) and has the highest estimated LTP.

In the particular simulation each forager has a k (= 50)-dimensional probabilistic term-frequency inverse document-frequency (PrTFIDF) text classifier [29], generated on a previously downloaded portion of the GeoCities database. Fifty clusters were created by Boley's clustering algorithm [30] from the downloaded documents. The PrTFIDF classifiers were trained on these clusters plus an additional one, the $(k + 1)^{th}$, representing general texts from the internet. The PrTFIDF outputs were non-linearly mapped to the interval (–1, +1) by a hyperbolic-tangent function. The classifier was applied to reduce the texts to a small dimensional representation. The output vector of the classifier for the page of URL A is **state(A)** = $(state(A)_1, ..., state(A)_k)$. (The $(k + 1)^{th}$ output was dismissed.) This output vector is stored for each URL.

A linear function approximator is used for LTP estimation. It encompasses k parameters, the *weight vector* **weight** = $(weight_1, ..., weight_k)$. The LTP of document of URL A is estimated as the scalar product of **state(A)** and **weight**: $value(A) = \sum_{i=1}^{k} weight_i\ state(A)_i$. During URL ordering the URL with highest LTP estimation is selected [for more details, see 21].

The weight vector of each forager is tuned by temporal difference learning [27, 28, 31]. Let us denote the current URL by URL_n, the next URL to be visited by URL_{n+1}, the output of the classifier for URL_j by **state(URL_j)** and the estimated LTP of a URL URL_j by $value(URL_j) = \sum_{i=1}^{k} weight_i\ state(URL_j)_i$. Assume that leaving URL_n to URL_{n+1} the immediate profit is r_{n+1}. Our estimation is perfect if $value(URL_n) = value(URL_{n+1}) + r_{n+1}$. Future profits are typically discounted in such estimations as $value(URL_n) = \gamma value(URL_{n+1}) + r_{n+1}$, where $0 < \gamma < 1$. The error of value estimation is

$$\delta(n, n+1) = r_{n+1} + \gamma value(URL_{n+1}) - value(URL_n).$$

We used throughout the simulations $\gamma = 0.9$. For each step $URL_n \rightarrow URL_{n+1}$ the weights of the value function were tuned to decrease the error of value estimation based on the received immediate profit r_{n+1}. The $\delta(n, n+1)$ estimation error was used to correct the parameters. The i^{th} component of the weight vector, $weight_i$, was corrected by

$$\Delta weight_i = \alpha\ \delta(n, n+1)\ state(URL_n)_i$$

with $\alpha = 0.1$ and $i = 1, ..., k$. These modified weights would improve value estimation in stationary and observable environments [see, e.g. 6 and references therein], but were also found efficient in large Web environments [21].

Without the RL-based URL ordering update algorithm the weight vector remains the same throughout the search.

Document Relevancy

A document or page is possibly relevant for a forager if it is not older than 24 h and the forager has not marked it previously. The selected documents are sent to the RA for further evaluation.

Multiplication of a Forager

During multiplication the weblog is randomly divided into two equal-sized parts (one for the original and one for the new forager). The parameters of the URL ordering algorithm (the weight vector of the value estimation) are either copied or new random parameters are generated. If

SFSW environment while for the other three environments the values are almost the same for the different type of foragers. RL and Fix foragers can get trapped in clustered environments as they cannot find outbound links in less relevant documents. The weblog algorithm provides a way to reach the best clusters that have been found and so escape the worst cluster. As such, the RL and Fix foragers in the SFSW environment perform worse than WL and WR foragers.

Finding new documents is hardest in the SFRandom environment for all foragers as incoming link degree distribution is exponential, which means that there are relatively more small degree pages than in the other three scale-free environments. It is thus more difficult for foragers to reach those pages that have lots of links to many pages. Overall, finding relevant documents is easiest and download efficiency is best in the SFSW environment for all foragers. This is because of the high number of found relevant documents compared to the other three environments. However, finding new URLs is hardest in the SFSW environment because of its clustered nature. It seems that selection fits the SFSW structure and vice versa, which the researchers suggest could explain the abundance of emergent SFSW structures in nature and perhaps even the organic nature of the web itself.

This finding can be understood in terms of the no free lunch theorem by considering it in reverse. If we find that a particular algorithm is more efficient than random search or than any other algorithm, then this winning algorithm 'knows' (or fits) the underlying problem better than the others. The researchers conclude from this that highly clustered SFSW structures and simple selective learning algorithms intrinsically match each other.

Consider data mining on the web. If the goal is to harvest the largest number of novel documents within the shortest time interval then, according to this study, the

the forager has a URL ordering update algorithm then the parameters are copied. If the forager does not have any URL ordering update algorithm then new random parameters are generated.

3.2 Reinforcing Agent

A RA controls the 'life' of foragers. It can start, stop, multiply or delete foragers. RA receives the URLs of documents selected by the foragers, and responds with reinforcements for the received URLs. The response is 100 arbitrary units (a.u.) for a relevant document and –1 a.u. for a not relevant document. A document is relevant if it has not yet been seen by the RA and it is not older than 24 h. The RA maintains the score of each forager working for it. Initially each forager has 100 a.u. score. When a forager sends a URL to the RA, the forager's score is decreased by 0.05. After each relevant page sent by the forager, the forager's score is increased by 1.

When the forager's score reaches 200 and the number of foragers is smaller than 16 then the forager is multiplied, that is a new forager is created with the same algorithms as the original one has, but with slightly different parameters. When the forager's score goes below 0 and the number of foragers is larger than 2 then the forager is deleted. Note that a forager can be multiplied or deleted immediately after it has been stopped by the RA and before the next forager is activated.

Foragers on the same computer are working in time slices one after the other. Each forager works for a certain amount of time determined by the RA. Then the RA stops that forager and starts the next one selected by the RA.

3.3 Foragers

A forager is initialized with parameters defining the URL ordering, and either with a weblog or with a seed of URLs. After its initialization a forager crawls in search paths, that is after a given number of steps the search restarts and the steps between two restarts form a path. During each path the forager takes 100 steps, i.e., selects the next URL to be visited with a URL ordering algorithm. At the beginning of a path a URL is selected randomly from the starting URL list. This list is formed from the 10 first URLs of the weblog. The weblog contains the possibly good starting URLs with their associated weblog values in descending order. The weblog algorithm modifies the weblog, thus the starting URL list before a new path is started. When a forager is restarted by the RA, after the RA has stopped it, the forager continues from the internal state in which it was stopped.

The URL ordering algorithm selects a URL to be the next step from the frontier URL set. The selected URL is removed from the frontier and added to the visited URL set to avoid loops. After downloading the pages, only those URLs (linked from the visited URL) are added to the frontier which are not in the visited set.

In each step the forager downloads the page of the selected URL and all of the pages linked from the page of selected URL. It sends the URLs of the possibly relevant pages to the RA. The forager receives reinforcements on any previously sent but not yet reinforced URLs and calls the URL ordering update algorithm with the received reinforcements.

4 Experiments

We conducted an 18-day-long experiment on the Web to gather realistic data. We used the gathered data in simulations to compare the weblog update (section 3.1) and RL algorithms (section 3.1). In the Web experiment we used a fleet of foragers using combination of RL and weblog update algorithms to eliminate possible biases on the gathered data. First, we describe the experiment on the Web, then the simulations. We analyze our results in the next section.

best approach is to use the selective weblog algorithm, provided that the domain of interest is SFSW. The scenario may be very different, however, if the goal is simply to download all novel documents. This is an intriguing problem that calls for the examination of the graphs of the not-downloaded documents for the different algorithms and, possibly, for the examination of the virtual graph between the foragers, explains Lőrincz. In this virtual graph, two foragers are connected to each other if they have sent back the same document. The connection is strong (they are close to each other) if they send back a large number of identical documents. Foraging may be optimized if the foragers learn which foragers are close to them and share the knowledge about their recent paths. Distributed communication between close foragers may improve data mining without too much communication overhead. This issue is the subject of the current research efforts of Lőrincz and colleagues.

David Bradley of Sciencebase.com

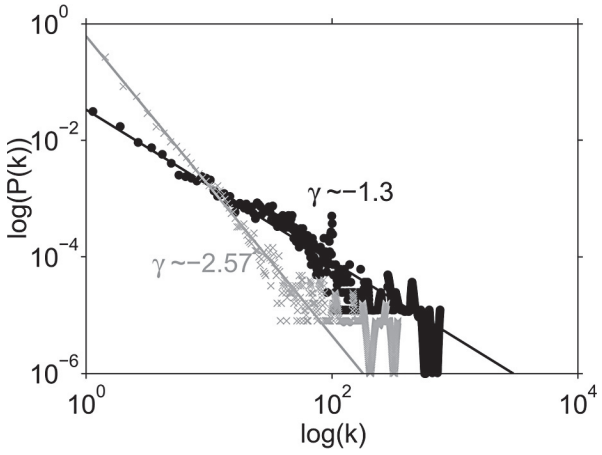

Fig. 1. Scale-free property of the internet domain. Log-log scale distribution of the number of (incoming and outgoing) links of all URLs found during the time course of investigation. Horizontal axis represents number of edges (log k), vertical axis represents relative frequency of a number of edges at different URLs (log $P(k)$). Dots and dark line correspond to outgoing links, crosses and gray line correspond to incoming links.

4.1 Data Collection on the Web

We ran the experiment on the Web on a single personal computer with Celeron 1,000 MHz processor and 512 MB RAM. We implemented the forager architecture (described in section 3) in Java programming language.

In this experiment a fixed number of foragers were competing with each other to collect news at the CNN web site. The foragers were running in equal time intervals in a predefined order. Each forager had a 3-min time interval and after that interval the forager was allowed to finish the step started before the end of the time interval. We deployed 8 foragers using the weblog update and the RL-based URL ordering update algorithms (8 WR foragers). We also deployed 8 other foragers using the weblog update algorithm but without RL (8 WL foragers). The predefined order of foragers was the following: 8 WR foragers were followed by the 8 WL foragers.

We investigated the link structure of the gathered Web pages. As shown in figure 1 the links have a power-law distribution ($P(k) = k^{\gamma}$) with $\gamma = -1.3$ for outgoing links and $\gamma = -2.57$ for incoming links, that is the link structure has the scale-free property. The clustering coefficient [32] of the link structure is 0.02 and the diameter of the graph is 7.2893. We applied two different random permutations to the origin and to the endpoint of the links, keeping the edge distribution unchanged but randomly rewiring the links. The new graph had 0.003 clustering coefficient and 8.2163 diameter, that is the clustering coefficient was smaller than the original value by an order of magnitude, but the diameter is almost the same. Therefore, we can conclude that the links of gathered pages form an SFSW structure.

The data storage for simulation is a central issue in our experiments. Pages are stored with 2 indices (and time stamps). One index is the URL index, the other is the page index. Multiple pages can have the same URL index if they were downloaded from the same URL. The page index uniquely identifies a page content and the URL from where the page was downloaded. For any foragers, at each page download we stored the following (with a time stamp containing the time of page download):

1 if the page is relevant according to the RA then store 'relevant'

2 if the page is from a new URL then store the new URL with a new URL index and the page's state vector with a new page index

3 if the content of the page is changed since the last download then store the page's state vector with a new page index but keep the URL index

4 in both previous cases store the links of the page as links to page indices of the linked pages

if a linked page is from a new URL then store the new URL with a new URL index and the linked page's state vector with a new page index

if the content of the linked page is changed since the last check then store the page's state vector with a new page index but the same URL index.

4.2 Simulations

For the simulations we implemented the forager architecture in Matlab. The foragers were simulated as if they were running on one computer as described in the previous section.

Simulation Specification

During simulations we used the Web pages that we gathered previously to generate different environments [note that the links of pages point to local pages (not to pages on the Web) since a link was stored as a link to a local page index]:

- Simulated documents had the same state vector representation for URL ordering as the real pages had.
- Simulated relevant documents were the same as the relevant documents on the Web.
- Pages and links appeared at the same (relative) time when they were found in the

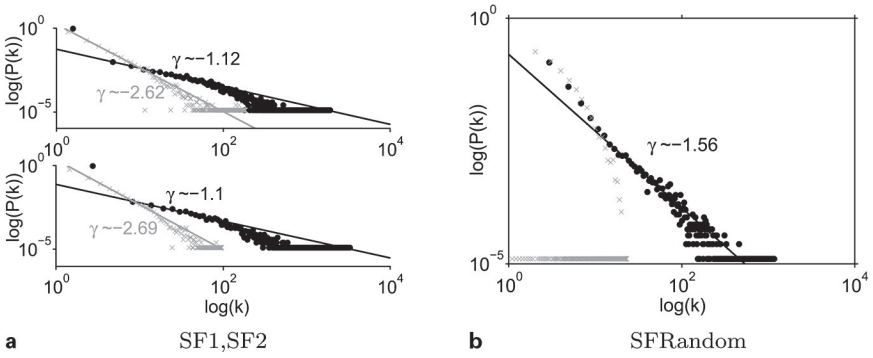

Fig. 2. Degree distribution of the environments. Dots and dark line correspond to outgoing link distribution. Crosses and gray line correspond to incoming link distribution. **a** Degree distributions of SF1 environment (upper part) and degree distributions of SF2 environment (lower part). **b** Degree distributions of SFRandom environment.

Web experiment – using the new URL indices and their time stamps.

- Pages and links are refreshed or changed at the same relative time as the changes were detected in the Web experiment – using the new page indices for existing URL indices and their time stamps.
- Simulated time of a page download was the average download time of a real page during the Web experiment.

We generated 4 different environments for the simulations:

1 *SFSW:* Each simulated page had exactly the same links as the original page had on the Web (a simulated page linked those simulated pages, page indices of those pages, which were linked by the original Web page).

2 *SF1:* In each second the new simulated pages had the same number of links as the original pages on the Web. A new simulated page was linked to simulated pages selected by the preferential attachment algorithm from the existing simulated pages.

3 *SF2:* The previous algorithm applied for the SF1 environment.

4 *SFRandom:* Similar to the SF1, but the linked simulated pages are selected from a uniform distribution of the pages.

The SFSW environment has exactly the same SFSW properties as the web environment downloaded by the web foragers. The SF1 and SF2 environments have 10 times smaller clustering coefficients than the SFSW environment has. These environments have scale-free degree distributions, although those are slightly different from the web environment (see fig. 2a).

The SFRandom environment also has 10 times smaller clustering coefficients than SFSW. The SFRandom environment has scale-free outgoing link degree distribution. But because of the uniform selection of linked documents the incoming link degree distribution is exponential (see fig. 2b). With the above given constraints this environment is the most random in the sense that all of the free parameters (linked documents) were selected from the uniform random distribution.

We conducted simulations with four different kinds of foragers in each environment:

1 WR foragers used both the weblog update and the RL-based URL ordering update algorithms.

2 WL foragers used only the weblog update algorithm without URL ordering update. Each WL forager had a different weight vector for URL value estimation – during multiplication the new forager got a new random weight vector.

3 RL foragers used only the RL-based URL ordering update algorithm without the weblog update algorithm. RL foragers had the same weblog with the first 10 URLs of the gathered pages – that is the starting URL of the Web experiment and the first 9 visited URLs during that experiment.

4 Fix foragers did not use the weblog update and the RL-based URL ordering update algorithms. These foragers had fixed starting URLs and fixed weight vectors, but the latter was different for each Fixed forager.

In each case, initially there were 2 foragers and they were allowed to multiply until reaching a population of 16 foragers. The simulation for each type of foragers was repeated 3 times with different initial weight vectors for each forager. The variance of the results show that there is only a small difference between simulations using the same kind of foragers, even if the foragers were started with different random weight vectors in each simulation.

Simulation Measurements

The first thing that we should note concerns the efficiency as a function of the number of crawlers. On a single computer, under the time sharing method we applied, and without direct competition between the different crawlers, we found that bipartition gives rise to a transient decrease of the efficiency of the new crawlers, but it quickly recovers. Within the limits of the number of crawlers that we studied (between 2 and 24), performance of the fleet is a slowly increasing function of the number of crawlers. We fixed the number of the crawlers and this slow dependence did not

Table 1. Investigated parameters

Downloaded	number of downloaded documents
Sent	number of documents sent to the RA
Relevant	number of found relevant documents
Found URLs	number of found URLs
Download efficiency	ratio of relevant to downloaded documents in 3-hour time window throughout the simulation
Sent efficiency	ratio of relevant to sent documents in 3-hour time window throughout the simulation
Exploration	ratio of found URLs to downloaded at the end of the simulation
Freshness	ratio of the number of current found relevant documents and the number of all found relevant documents [25]; a stored document is current, up-to-date, if its content is exactly the same as the content of the corresponding URL in the environment
Age	a stored current document has 0 age, the age of an obsolete page is the time since the last refresh of the page on the Web [25]

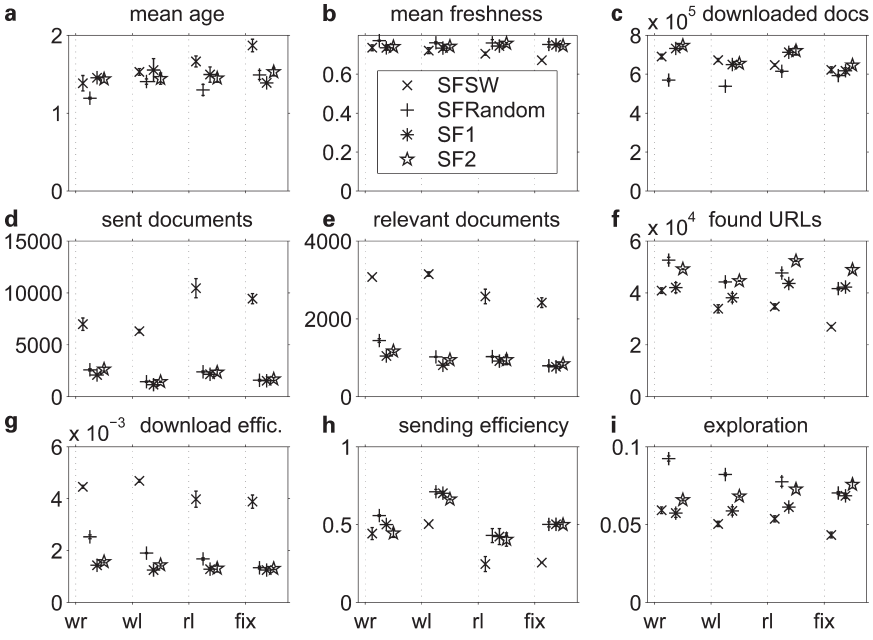

Fig. 3. Measured parameter values. The investigated parameters are shown in **a–i**. Each part of the figure contains the parameter values for the four types of foragers in the four columns as shown along the bottom x-axis. The 4 different markers correspond to the measured parameter values in the 4 environments as shown in **b**. On each marker an error bar shows the standard deviation of the corresponding parameter values for the 3 simulations. **a** Mean age is in hours.

enter our considerations. Table 1 shows the investigated parameters during simulations.

The parameter 'download efficiency' is relevant for the site where the foragers should be deployed to gather the new information. The parameter 'sent efficiency' is relevant for the RA. Note that during simulations we are able to immediately and precisely calculate freshness and age values. In a real Web experiment this is impossible, because of the time needed to download and compare the contents of all of the real Web pages to the stored ones.

5 Discussion

We observed that the efficiency of the algorithms depends strongly on the weight vectors. As we have mentioned above, the number of foragers had slight effects on the efficiency. This observation is supported by the fact that upon bipartition the weight vectors of the descendant foragers are similar, causing the descendant foragers to follow similar paths and to spoil the performance of each other for a while. This is the reason that efficiency shows a transient decrease, but as a result of adaptation it quickly disappears.

The dependence of the efficiency on the number of starting points is not too much different. Two foragers could use 20 different starting points, whereas 16 foragers could use 160 different starting points, but the efficiency was only slightly influenced by this order of magnitude difference. That is, the number of starting points and the number of foragers are less important than the weight vectors in our simulations indicating that 20 starting points or so, which can adapt, can efficiently shatter the structure. Note that other parameters, e.g., larger number of crawlers and more than one computer can change this simple picture. The weak dependence, however, is most advantageous for our purpose namely to be confident that we can study structural dependencies.

The measured parameter values are presented in figure 3. This figure contains the values for each measured parameter for each type of forager in each type of environment. From the subfigures we can conclude the following.

It can be seen in figure 3a and b that freshness and age values of different foragers are changing in the SFSW environment (marker ×) while in the other 3 environments the values are almost the same for the different type of foragers. RL and Fix foragers can get trapped in clustered environments, and cannot easily escape (find outgoing links leaving the cluster) from clusters containing less relevant documents. The weblog algorithm provides a way to go to the best clusters found and in this way to escape from the worse clusters. This can be the reason why the RL and Fix foragers in the SFSW environment perform worse than WL and WR foragers.

In figure 3c it can be seen that finding new documents is the hardest in the SFRandom environment for all foragers. In this environment the pages have exponential incoming link degree distribution, which means that there are relatively more small degree pages than in the other 3 scale-free environments. It is harder to get to pages which link to many pages. Therefore, it is harder for the foragers to find documents which have not yet been seen.

It can be seen in figure 3d and e that finding relevant or possibly relevant documents is the easiest in the SFSW environment for all foragers. This environment is more clustered than the other 3 environments. When a forager finds a relevant document in a cluster then it finds other relevant documents in that cluster while it cannot escape from the cluster. The other 3 environments are less clustered. Foragers can get out from clusters more easily by following the links and therefore having to forage the entire environment for new relevant documents.

In figure 3g it can be seen that the download efficiency is the best in the SFSW environment for all foragers. This is because of the high number of relevant documents found compared to the other three environments.

In figure 3h it can be seen that the sent efficiency of the WR forager is the same in all environments, although these foragers found less relevant documents in not SFSW environments but also sent back less documents to the RA. The other 3 foragers sent more or less documents to the RA in the not SFSW environments; therefore, their sent efficiencies are the worst in the SFSW environment, although these foragers also found the most relevant documents in the SFSW environment compared to the other 3 environments.

It can be seen in figure 3f that finding new URLs is the hardest in the SFSW environment because of its clustered nature. The foragers check the same URLs for changed documents in the clusters; therefore, they can collect many new relevant documents. In the other 3 environments foragers do not get trapped as much in the clusters and they search the whole environment continuously.

Now, consider table 2, which contains some of the data of figure 3. The number of sent relevant documents is somewhat larger for the WL foragers than for WR foragers in SFSW environments (3,149 and 3,075, respectively, i.e., the difference is about 2.3%). This slight difference changes sign and becomes much more pronounced in all other environments. For example, in the SFRandom environment the numbers are 1,029 and 1,441, i.e., the difference is about 28%, but in the opposite direction. It is important to note that, if these foragers compete with each other, then the slight 2.3% difference or the larger 28% difference both enter the argument of an exponential because of bipartitions. Thus, even slight differences can become large, and may give rise to overwhelming population differences. Such competitive runs are under way. The issue becomes more pronounced for real situations, where time is not shared on a single computer and all foragers may search for food at all times.

Our results and the results of Annunziato et al. [4] cannot be compared directly. The most obvious reason is that their investigations were restricted to SF, SW and random structures, but they did not study the SFSW structure, which plays a central role in our work. Artificial studies are, however, desirable, because in such examples one can finely gauge the different components and may find the necessary and sufficient conditions of our findings.

We consider the following findings important: Bipartition gives rise to certain transient disadvantages if the descendants are similar. They will have to share the food until they start to learn. Still, selective learning can be more effective than selective learning combined with other methods, or than other methods alone if the environment is SFSW. It seems that selection fits the SFSW structure and vice versa. This

Table 2. Quantitative results for algorithm-structure pairs

	Weblog		Weblog and RL	RL	
	SFSW	SFR	SFSW	SFSW	SFR
Number of sent documents	6,313	1,443	6,985	10,455	2,396
Number of relevant documents	3,149	1,023	3,075	2,575	1,029
Sending efficiency	0.5035	0.7106	0.4425	0.2463	0.4299
Number of found URLs	33,882	44,217	40,888	34,759	47,668

could be a good reason for the abundance of emergent SFSW structures in nature. This can be understood through the no free lunch theorem if it is read backwards: if we find that a particular algorithm is more efficient than random search or than any other algorithm, then this winning algorithm 'knows' (fits) the underlying problem best among the investigated algorithms. Based on our computer simulations it seems that highly clustered SFSW structures and simple selective learning algorithms match each other.

6 Conclusions

We investigated algorithms using evolutionary, RL, and combined evolutionary and RL strategies. We experimented with environments that have different degree distributions and clustering coefficients. We generated different topological environments, using data collected during real Web search. Our study focused on the task of searching for new relevant documents. We found that in the SFSW environment the evolutionary weblog update algorithm performs the best. It outperformed a RL-based algorithm and the combinations of these two algorithms. We conjecture that the highly clustered nature and the small diameter of the environment match simple selection over other more sophisticated learning schemes. However, when the scale-free nature of the environment was kept but the small diameter of the environment was increased simply by restructuring the environment, then other algorithms performed better than the simple selection. We found in the 3 not small world environments that the combination of the weblog and RL algorithms are the best. That is, when the diameter of the world becomes larger, then estimation of the long-term cumulated reward becomes important. Moreover, the combined algorithm showed the smallest performance variation both on the SFSW and on scale-free environments.

Acknowledgements

This material is based upon work supported by the European Office of Aerospace Research and Development, Air Force Office of Scientific Research, Air Force Research Laboratory, under Contract No. FA8655-03-1-3036. This work is also supported by the National Science Foundation under grants No. INT-0304904 and No. IIS-0237782. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the European Office of Aerospace Research and Development, Air Force Office of Scientific Research, Air Force Research Laboratory.

References

1 Wolpert DH, Macready WG: No free lunch theorems for optimization. IEEE Trans Evolutionary Comput 1997; 1: 67–82.
2 Barabási A, Albert R, Jeong H: Scale-free characteristics of random networks: the topology of the world wide web. Physica A 2000; 281: 69–77.
3 Albert R, Barabási A: Statistical mechanics of complex networks. Rev Mod Phys 2002; 74: 47–91.
4 Annunziato M, Huerta R, Lucchetti M, Tsimring LS: Artificial life optimization over complex networks. 4th International ICSC Symposium on Engineering of Intelligent Systems, Funchal, 2004.
5 Kleinberg J, Lawrence S: The structure of the web. Science 2001; 294: 1849–1850.
6 Sutton R, Barto A: Reinforcement Learning: an Introduction. Cambridge, MIT Press, 1998.
7 Eiben AE, Smith JE: Introduction to Evolutionary Computing. Berlin, Springer, 2003.
8 Fryxell J, Lundberg P: Individual Behavior and Community Dynamics. London, Chapman & Hall, 1998.
9 Clark C, Mangel M: Dynamic State Variable Models in Ecology: Methods and Applications. Oxford, Oxford University Press, 2000.
10 Kampis G: Self-Modifying Systems in Biology and Cognitive Science: a New Framework for Dynamics, Information and Complexity. Oxford, Pergamon, 1991.
11 Csányi V: Evolutionary Systems and Society: a General Theory of Life, Mind, and Culture. Durham, Duke University Press, 1989.
12 Pachepsky E, Taylor T, Jones S: Mutualism promotes diversity and stability in a simple artificial ecosystem. Artif Life 2002; 8/1: 5–24.
13 Tesauro GJ: Temporal difference learning and td-gammon. Commun ACM 1995; 38: 58–68.
14 Mataric MJ: Reinforcement learning in the multi-robot domain. Autonomous Robots 1997; 4/1: 73–83.
15 Stafylopatis A, Blekas K: Autonomous vehicle navigation using evolutionary reinforcement learning. Eur J Operational Res 1998; 108: 306–318.
16 Moriarty D, Schultz A, Grefenstette J: Evolutionary algorithms for reinforcement learning. J Artif Intell Res 1999; 11: 199–229.
17 Tuyls K, Heytens D, Nowe A, Manderick B: Extended replicator dynamics as a key to reinforcement learning in multi-agent systems; in Lavrac N, et al (eds): ECML 2003, LNAI 2837. Berlin, Springer, 2003, pp 421–431.
18 Kondo T, Ito K: A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control. Robot Auton Syst 2004; 46: 11–124.
19 Kökai I, Lörincz A: Fast adapting value estimation based hybrid architecture for searching the world-wide web. Appl Soft Comput 2002; 2: 11–23.
20 Lörincz A, Kókai I, Meretei A: Intelligent high-performance crawlers used to reveal topic-specific structure of the WWW. Int J Founds Comp Sci 2002; 13: 477–495.
21 Palotai Z, Gábor B, Lörincz A: Adaptive highlighting of links to assist surfing on the internet. Int J Inf Technol Decis Making 2005; 4: 117–139.
22 Angkawattanawit N, Rungsawang A: Learnable topic-specific web crawler; in Abraham A, Ruiz-del-Solar J, Köppen M (eds): Hybrid Intelligent Systems. Amsterdam, IOS Press, 2002, pp 573–582.
23 Menczer F: Complementing search engines with online web mining agents. Decis Support Syst 2003; 35: 195–212.
24 Risvik KM, Michelsen R: Search engines and web dynamics. Comput Networks 2002; 32: 289–302.
25 Cho J, Garcia-Molina H: Effective page refresh policies for web crawlers. ACM Trans Database Syst 2003; 28: 390–426.
26 Edwards J, McCurley K, Tomlin J: An adaptive model for optimizing performance of an incremental web crawler. Proceedings of the 10th International Conference on World Wide Web, Hong Kong, 2001, pp 106–113.
27 Szita I, Lörincz A: Kalman filter control embedded into the reinforcement learning framework. Neural Comput 2004; 16: 491–499.
28 Schultz W: Multiple reward systems in the brain. Nat Rev Neurosci 2000; 1: 199–207.
29 Joachims T: A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization; in Fisher DH (ed): Proceedings of ICML-97, 14th International Conference on Machine Learning (Nashville, US, 1997). San Francisco, Morgan Kaufmann, 1997, pp 143–151.
30 Boley D: Principal direction division partitioning. Data Mining Knowledge Discovery 1998; 2: 325–344.
31 Sutton R: Learning to predict by the method of temporal differences. Mach Learn 1988; 3: 9–44.
32 Watts DJ, Strogatz SH: Collective dynamics of 'small-world' networks. Nature 1998; 393: 440–442.