On How Ants Put Advertisements on the Web

Tony White, Amirali Salehi-Abari, and Braden Box

School of Computer Science, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6 Canada
{arpwhite,asabari,bbox}@scs.carleton.ca

Abstract. Advertising is an important aspect of the Web as many services rely on it for continued viability. This paper provides insight into the effectiveness of using ant-inspired algorithms to solve the problem of Internet advertising. The paper is motivated by the success of collaborative filtering systems and the success of ant-inspired systems in solving data mining and complex classification problems. Using the vector space formalism, a model is proposed that learns to associate ads with pages with no prior knowledge of users' interests. The model uses historical data from users' click-through patterns in order to improve associations. A test bed and experimental methodology is described, and the proposed model evaluated using simulation. The reported results clearly show that significant improvements in ad association performance are achievable.

Keywords: Ant Colony Optimization, Stigmergy, Pheromone, Collaborative Filtering.

1 Introduction

The proliferation of content on the World Wide Web (WWW) in the past decade has been, in the most part, economically supported by web advertising. A web advertisement is a section of a web page that is dedicated not to the content of the page but to graphics and/or text promoting the product or service of a third party. A web advertisement usually offers a link to an outside web site that provides further details regarding the product. The advertising party and the webmaster who owns the content have a contract which states that the advertiser will pay the webmaster in exchange for this service. The precise amount offered is frequently conditional on the success of the ad in compelling the end user to follow its link for more information. This success is usually measured by click-through rate (or CTR). A CTR is calculated by dividing the number of users who clicked on a specific ad by the number of times the ad is delivered.

It is then in the webmaster's best financial interest to maximize the number of his users that will, in fact, click-through. One approach to doing this is to ensure that the ad's content is matched to the users' interests, if there is a choice amongst candidate ads for a web page. It is this problem of choice that motivates the research reported in this paper. The anonymity of the web creates several obstacles to effectively resolving this problem of choice. It is not possible to track

N. García-Pedrajas et al. (Eds.): IEA/AIE 2010, Part II, LNAI 6097, pp. 494–503, 2010.

who exactly is coming to your website without browser cookies or some other invasive technology, and even these are unreliable[1]. Even if it could be known which user is which, it is not easy to gauge their interests and demographics outside broad categories like geographic location or browser usage. A webmaster can only rely on one piece of information from the user when choosing which ad to match with a served page, and that is the content of the requested page.

Despite the limited amount of information, the webmaster must determine the users' interests, or at least which ads they are most likely to click on. Our main contribution includes the introduction of an ant-based algorithm for the association of web pages and ads with each other. Utilizing stigmergic [2,3] principles, the proposed algorithm builds models that can be quickly updated and provide recommendations for ad-serving. Moreover, we introduce a simulation test bed for evaluation of the proposed algorithm. The experiments performed demonstrate the utility of the proposed algorithm.

The paper consists of 6 further sections. The paper continues by providing important background information on Ant Colony algorithms in Section 2. Section 3 briefly describes related work in the area of ad association using biologically-inspired algorithms. Section 4 describes the main contributions of this paper: algorithms for ad association and a test bed that is used to evaluate them. Sections 5 and 6 describe the experimental setup and results respectively. Section 7 summarizes the key messages of the paper and briefly highlights potential future work.

2 Background

The heuristics that an ant colony uses to find food have inspired a computational metaheuristic that is known as Ant Colony Optimization (ACO) [4]. Starting with a simple, connected graph with start and destination nodes and every edge having a pheromone level, τ_{ij}, each ant steps from the node it is on to another connected node. The probability of the selection of an edge, e_{ij}, to follow at time t while the ant is located at node i can be calculated using Equation 1. Here, τ_{ij} is the amount of pheromone on edge e_{ij} and η_{ij} represents the desirability of a given direction. $N(i)$ contains the neighbors of node i. The parameters α and β are system parameters.

$$P_{ij}(t) = \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}}{\sum_{x \in N(i)} \tau_{ix}^{\alpha}(t)\eta_{ix}^{\beta}} \tag{1}$$

This process is repeated with each ant at each node until it reaches the destination. When the destination is reached an amount of pheromone is deposited on each edge that is inversely proportional to the total length of the path. To prevent premature convergence, pheromones are allowed to evaporate over time. Algorithmically, this means at each iteration, reduce the pheromone level, τ_{ij}, by multiplying it by $(1 - \rho)$ where $0 < \rho < 1$ is the evaporation rate. With this addition, we get the simplest form of ACO [4]. An ACO variant will be used to recommend Web advertisements and will be detailed in Section 4.1.

3 Related Work

The algorithms used for ad selection for Web pages are proprietary and remain largely unreported in the literature. Furthermore, *real time ad selection* is not based upon online click-though, which is the motivation for the research reported in this paper. However, the value of click-through data in this domain is well understood. See, for example, [5,6].

The problem of offline ad association can be viewed as a data mining problem if a large body of page requests can be used to induce classifiers. While the body of literature is large in this general space, Kim et al. [7] have used decision trees to guide the creation of advertisements for online storefronts and [6] has optimized search results using support vector machines and click-through data. We strongly believe that [5] could be used to analyze ads provided to our system to create the *adKeywords* vectors shown in the *clickThrough* algorithm (see Section 4) and facilitate the use of non-zero values of β (see Algorithm 3 and Table 1). However, our interest in this paper is the incremental creation of classifiers online and, more specifically, through a use of biologically-inspired algorithms. With this latter qualification, prior research is sparse, with *AdPalette* [8] being noteworthy. AdPalette uses genetic algorithms to customize advertisements with usage, relying on crossover and mutation in order to combine promising ad components on pages.

4 Model

The research reported here was performed in a simulation environment. Figure 1 represents the actual Web environment being simulated. Simulated users with defined preferences create queries that are used to generate responses that contain a simulated advertisement. In the Web environment shown in Figure 1, users (e.g., Bob and Alice) interact with one or more web servers (e.g., Web Server A and B). When Bob asks for a page from Web Server A (indicated by 1 in the circle) content is returned that contains JavaScript that runs inside of Bob's browser. The JavaScript causes the Ad Server to be contacted with keywords extracted from the content delivered by Web Server A. The ad returned from the Ad Server is shown by the number 3 in a circle in Figure 1.

The simulation models the interaction that occurs between the web page users and the web server that processes page requests and matches the advertisements to the content. Two types of processes run simultaneously: one server script where most of the actual computing occurs, and a *user script* that contains randomly generated users that create and send off page requests to the server to be processed. Essentially, the user script models a user's interactions with various web servers that are connected, in turn, to an ad serving system. The functions of the user script are to (a) generate users and (b) generate queries and assess responses. The *server script* represents the functions of the ad serving system. The server script is responsible for analyzing the queries that it receives and making a decision as to which ad to serve.

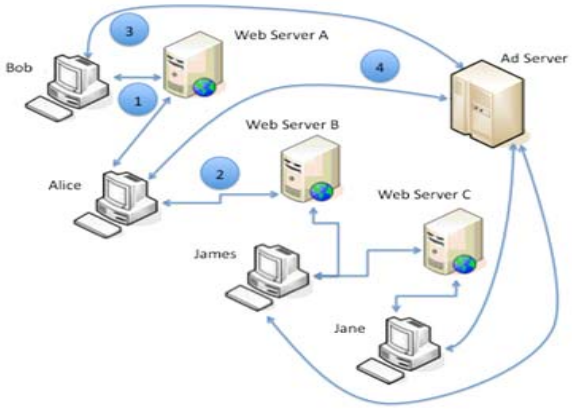

Fig. 1. The Ad Placement Scenario

The first function of the user script is to generate up to l users. Each user entity is meant to simulate one actual user's preferences and choices. To represent the variability of users' interests, each user has a rating for each of a pool of m keywords that represents how interested the user is in that particular subject. Note, since we are merely simulating a user, actual keywords are not used but the preferences, *userInterest*, recorded as a vector of values $[k_0, k_1, \ldots, k_{m-1}]$, with the value of each k_i being between 0 – meaning no interest in the subject i – and 1 – meaning the highest possible interest in i – are instead. These values are randomly generated, but are weighted towards the extremes such that there is a one third chance of the interest level being between 0.75 and 1, and a one third chance of the interest level being between 0 and 0.25. This is meant to represent that is more likely that a person has a considerable amount of interest in a topic or very little interest at all, as opposed to being ambivalent about it.

After the user is created, the user script synthesizes finding a page of interest, defined by keywords, to the user. Pages are represented by randomly generating a relevance vector $[r_0, r_1, \ldots, r_{m-1}]$ to represent a possible page, again with values between 0 and 1 that represent how tightly tied to the subject matter the content is. In an actual implementation with real Web content, this could be determined using a vector space model [9] that analyzes the page content relevant to other pages. Since users do not randomly surf pages, but go to pages that match their interests, only pages with keywords relevant to the user are of interest. To simulate this, the angle between the interest vector of the user and the relevance vector of the page is first determined using Equation 2.

$$angle(a, b) = \arccos\left(\frac{\sum_{i=0}^{m} a_i b_i}{\|a\|\|b\|}\right) \tag{2}$$

In Equation 2, a and b are vectors of size m with $\|a\|$ and $\|b\|$ being the lengths of the vectors a and b respectively. If the angle computed for the page is greater than some chosen threshold, R_{thresh}, it is rejected. If the page is rejected, then each of

Algorithm 1

generatePage(Array *userInterest*)
page = new Array[*m*]
for $i = 0$ to $m - 1$ **do**
  page[*i*] = random value from −0.25 to 1.25
  if *page*[*i*] > 1 **then**
    page[*i*] = *page*[*i*] - 0.25
  end if
  if *page*[*i*] < 0 **then**
    page[*i*] = *page*[*i*] + 0.25
  end if
end for
angle = angle between *page* and *userInterest* using Equation 2
while *angle* > R_{thresh} **do**
  for $i = 0$ to $m - 1$ **do**
    20% chance of: $page[i] = \frac{page[i]+userInterest[i]}{2}$
  end for
  angle = angle between *page* and *userInterest* using Equation 2
end while
return *page*

the m values has a 20% chance of being averaged with the user's interest values. This continues in a loop until the threshold is met. When the *generatePage* algorithm (see Algorithm 1) completes it is guaranteed that the generated page represents the interests of the modeled user; however, there may be content unrelated to the user's primary interests.

When the server receives a page from a user, its function is to decide which ad should be matched with the given page. This is achieved by using the Ad Association (A2) algorithm as defined in Section 4.1. Finally, the page will be sent back to the appropriate user with one of n possible ads attached. The user them evaluates the ad, and determines whether to click it. In a real-life scenario, the person receiving the ad would judge the ad on his own using preferences and goals, and choose whether the ad has piqued his interest enough to click it. However, it would be incredibly hard to simulate a human in this way, so it has been simplified as follows. Each ad has a number of keywords associated with it, denoted by pm, of the m possible keywords ($pm < m$). The more interest a user has in the pm keywords, the more likely the ad is to be a success. The chance of success is determined on a logarithmic scale. For example, with 10 keywords associated with an ad, there is an expected value of 1/100 th chance of a click-through, but varying from $1/10,000$ th chance if the user has no interest at all in the ad's keywords and guaranteed success if the user has maximum interest in all keywords. The *clickThrough* algorithm below indicates how success is computed, with the $adKeywords$ array containing 0 for completely irrelevant keywords not thought to be useful for the ad and 1 for keywords that are considered important or completely relevant. The $adKeywords$ array therefore contains pm entries that are 1 and $m - pm$ entries that are 0. The *normalization* value is a constant for the system.

Algorithm 2

clickThrough(Array $adKeywords$, Array $userInterest$)
$total = 0$
for $i = 0$ to $n - 1$ **do**
  if $adKeywords[i] == 1$ **then**
    $total = total + userInterest[i]$
  end if
end for
return $10^{(total-pm)*normalization}$

Finally, when the success of the ad is determined, information is once again passed back to the server. The server executes the A2 algorithm and makes changes based on the page relevance vector, the ad selected and the user's choice in regard to the ad. This whole process repeats many times; multiple users each making a series of page requests.

4.1 Ad Association (A2) Algorithm

In this ant-inspired algorithm, each ad is a given fixed path with m nodes, one for each of the possible interest keywords along with the $adKeywords$ described in the previous section. When an ad is served to the user, and is successful, then it positively reinforces that ad's path. If it fails, it negatively reinforces the path.

There are 3 parts to the A2 Algorithm: the model, a method for choosing the best ad considering a page input, and a method for changing the model. The model, M, is a collection of one vector per ad, a, and each vector has a value for each of the m interest keywords, i.e., M=$\{v_{01}, v_{02}, \ldots, v_{0m-1}, v_{11}, \ldots, v_{n-1m-1}\}$. Each of these nodes, v_{ij}, initially contains a value, τ_0, just slightly above zero representing the amount of pheromone on that part of the path. There are n ads that can be served. Furthermore, each ad has pm defining keywords, these being used to decide on whether a click-through occurs or not. The pm keywords are chosen from the m possible keywords.

As the server receives page requests from users, the A2 algorithm (See Algorithm 3) executes for the purpose of determining which of the n ads should be returned. The algorithm proceeds by comparing the page's relevance vector with each of the n ads by measuring the angle between the two vectors using Equation 2. This comparison has two components: first, the feedback gathered from previous ad associations and second, the comparison between the page and the ad keywords that are considered relevant. The smaller the angle, the more similar the ad's vector is to the relevance vector. The server chooses which ad to return randomly biased by rank. There is a $\frac{1}{2^r}$ chance that the r^{th} best ad is returned to the user, thus ensuring a heavy bias towards the best ads; i.e., the best ad will only be returned half of the time. Once the server sends out the ad, it waits for information on whether the user clicked the ad. If the ad was successful, the *recordAd* algorithm executes. This algorithm ensures that the vector corresponding to that successful ad, i, is updated so that it increases its pheromone values for all keywords in i, but increases the page's more relevant

Algorithm 3. Ad Association (A2) Algorithm

```
decideOnAd(Array page)
angles = new Array[n]
for a = 0 to n − 1 do
  angleToAd = angle between page and ads[a] using Equation 2
  angleToV = angle between page and v[a] using Equation 2
  angles[a] = (angleToAd^β) × (angleToV^α)
end for
angles = sortInIncreasingOrder(angles)
index = 0
while true do
  if index >= n then
    return last ad in angles array
  else
    50% chance of returning ad with angle at index
    Otherwise: index = index + 1
  end if
end while

recordAd(adNumber, Array page, success)
if success == true then
  for i = 0 to m − 1 do
    v[adNumber][i] = v[adNumber][i] + page[i] ∗ c
  end for
else
  for i = 0 to m − 1 do
    v[adNumber][i] = v[adNumber][i] − page[i] ∗ k
  end for
end if
```

keywords more, as shown in Equation 3. The relevance vector is r, and c is a constant that controls how much each success influences the model, $0 < c < 1$. This is equivalent to an ant spreading pheromone on the path to the successful ad, and making that path more appealing to future ants with similar vectors that match.

$$\forall j \quad v_{ij}^{t+1} = v_{ij}^{t} + r[j] \times c \tag{3}$$

To counter the pheromone values from growing out of control, values are bounded and the model also adjusts them when an ad fails. In a real-life application, it is much harder to tell when an ad fails as no message can be sent to the server saying that the user did not do some action. To circumvent this, it is assumed that there is a timeout function such that if the ad is not successful in a given time frame, then it counts as a failure. In the simulation, the user script just returns a "failure" result. When the server receives a failure, it reduces all pheromone values for the associated ad, i, but reduces it more for the keywords most relevant to the page in question. This effect is shown in Equation 4,

where k is a constant that controls how much each failure influences the model, $0 < k < 1$ and $k << c$. Here, k has the role of evaporation in ACO. τ_0 is a very small number, $\tau_0 << k$.

$$\forall j \quad v_{ij}^{t+1} = v_{ij}^{t} + max(r[j] \times k, \tau_0) \tag{4}$$

The purpose of having this pheromone reduction is to prevent one ad from being overused early on, and becoming dominant before other ads get a chance to demonstrate relevance. Equations 3 and 4 form the basis of the *recordAd* algorithm.

5 Experiments

Multiple trials were completed for the A2 algorithm to ensure consistency and result reproducibility. Three trials are reported in Figure 2 in order to show the typical variability in simulated performance.

Table 1. Experimental Parameters

Variable	**Value**	**Variable**	**Value**
R_{thresh}	20°	k	0.01
τ_0	10^{-6}	c	0.999
n	20	pm	10
m	100	l	100
$normalization$	0.4	α	1
Total page requests	$500,000$	β	0

The values of c and k were chosen to reflect the relative probabilities of success and failure respectively; the ratio being approximately 100 to 1 for the scenario modeled here. The value of β were chosen to reflect an extremely pessimistic scenario in which nothing was known about the keywords associated with the advertisements to be presented. In essence, this value of β says that we know nothing about the contents of the ad, which may be true if the ad is an image or video or the provider seeks to provide a "black box" ad. That said, this is an extreme scenario and was chosen in order to test whether the system could improve based purely upon observed feedback from users.

The value of the *normalization* coefficient was chosen to ensure that the average chance of click-through for a page with half of the expected keywords correct would be 0.01, with the range of probabilities varying from 1 (for all expected keywords correct) to 0.0001 (for no keywords correct).

The page requests were broken down into 50 blocks of $10,000$ requests each. For each block, the average success rate as determined by the user was measured. Also measured was the average expected value of the success rate if the ads were chosen randomly. This is shown as a horizontal line in Figure 2.

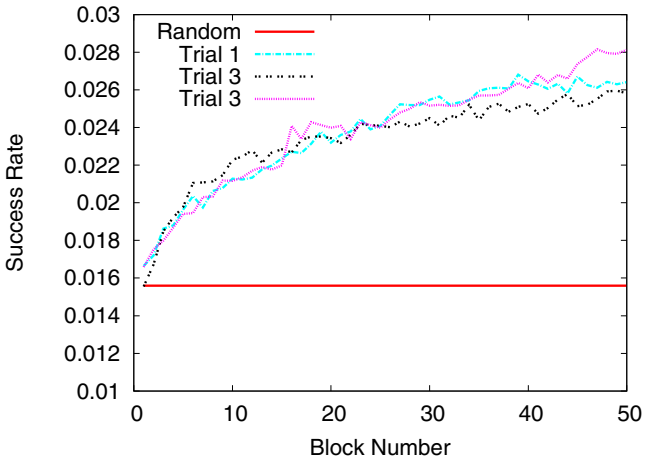

Fig. 2. Success Rate Variation with Block Number

6 Results and Discussion

The data gathered from testing the A2 Algorithm is promising. As shown in Figure 2 the average chance of ad success will be increased over time. Over the course of 500,000 page requests, there was a 70% to 80% improvement in efficiency over the random pairing of ads when viewed across the 3 trials reported here.

Note that, the proposed algorithm does not always return the best match from its model, but returns one from the top few ads with very high frequency (87.5% from the top 3 ads). This is important and deliberate. It was found during the implementation and associated experimentation that without this feature, the model would converge too quickly on one ad that happened to show promise early on. Success would breed more recommendations for that ad, which would only increase the likelihood of it being recommended again later. Adding the chance that any ad could be picked kept the model from being dominated too early by any particular ad. Thus, robustness was maintained.

As can be observed in Figure 2, the system continues to learn even after 500,000 page requests making it likely that 100% improvement over initial system performance is achievable. Combining the clear benefits that this algorithm produces in terms of advertising success and the ease with which it handles large quantities of data makes it attractive.

7 Conclusions

This paper provides insight into the effectiveness of using an ant-based algorithm to improve Internet advertising. A webmaster could use the proposed A2 algorithm with minor modifications. As shown with an artificial environment, it should be able to increase ad success, and therefore, advertising revenue by

over 70% after processing a reasonable amount of traffic for a large website. If it were used, with refinements made to adjust for real-life variables and efficiencies added to reflect issues unique to the website, it would be a simple piece of software with potential revenue benefits. It is worth looking further into using the A2 algorithm as a predictor of advertisement effectiveness. We believe that clickthrough data mining techniques as described in [5] and [6] (as examples) could provide a valuable starting point for system ad keyword initialization thereby allowing for better-than-random initial system performance. Furthermore, larger data sets should be tested beyond the small problem space that is used here.

Beyond this, the next stage would be to implement the algorithm on a real web server. It would run continuously and intercept real incoming customer data and produce actual ads, using vector-space models to convert the requested pages into the vectors used by the algorithm. It could be further refined to take into account that some advertisements may pay more to be shown, that some ads have different sizes and page positioning from others and that more than one ad is often shown on a site at once.

References

1. Kristol, D.M.: Http cookies: Standards, privacy, and politics. ACM Trans. Internet Technol. 1(2), 151–198 (2001)
2. Dorigo, M., Bonabeau, E., Theraulaz, G.: Ant algorithms and stigmergy. Future Gener. Comput. Syst. 16(9), 851–871 (2000)
3. Ramos, V., Abraham, A.: Swarms on continuous data. In: The 2003 Congress on Evolutionary Computation, pp. 1370–1375 (2003)
4. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Book (2004)
5. Joachims, T., Radlinski, F.: Search engines that learn from implicit feedback. Computer 40, 34–40 (2007)
6. Joachims, T.: Evaluating retrieval performance using clickthrough data. In: SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval (2002)
7. Kim, J.W., Lee, B.H., Shaw, M.J., Chang, H.L., Nelson, M.: Application of decision-tree induction techniques to personalized advertisements on internet storefronts. International Journal of Electronic Commerce 5(3), 45–62 (2001)
8. Karuga, G.G., Khraban, A.M., Nair, S.K., Rice, D.O.: Adpalette: an algorithm for customizing online advertisements on the fly. Decision Support Systems 32(2), 85–106 (2001)
9. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM 18(11), 613–620 (1975)