Web User Session Reconstruction Using Integer Programming

Robert F. Dell
Operations Research Department
Naval Postgraduate School
Monterey, California, USA
dell@nps.edu

Pablo E. Román*, Juan D. Velásquez
Department of Industrial Engineering
University of Chile
República 701 Santiago, Chile.
proman@ing.uchile.cl, jvelasqu@dii.uchile.cl

Abstract

Web usage mining has proven to be an important advance for e-business systems, by finding web user buying patterns and suggesting ways to improve web user navigation. An important input is web user sessions that must be reconstructed from web logs (sessionization) when such sessions are not otherwise identified. We present a new approach for sessionization based on an integer program. We compare results of our approach with the timeout heuristic on web logs from an academic web site. We find our integer program provides sessions that better match an expected empirical distribution with about a half of the standard error of the heuristic.

1. Introduction

A log file from a web server (*web log*) contains records of users' browsing activities. For marketing purposes, web logs are a potential large source of data ($\sim$ Gb) on customer preferences [14]. A web log [22] is typically a large text file with each line (*register*) containing the following: the time of a document (web page) access, the IP address of the user, the *agent* field that identifies the user's browser, and the document retrieved. A log file contains evidence of each web user's activities and serves as a huge electronic survey of a web site. This has motivated considerable research on how to mine this important information, a field coined Web Usage Mining [14].

A web log by itself does not necessarily reflect a sequence of an individual user's document access, it registers every retrieval action but without a unique identification for each user. This originates the need to reconstruct a user's session from the information available (*sessionization*). Prior work on sessionization has relied on heuristics [2, 6, 20]. These heuristics have been applied with success on a variety of studies that include web user navigational behavior, recommender systems, pattern extraction, and web site keyword analysis [14]. In this paper, we propose an integer program to construct sessions and demonstrate its advantages using web logs from an academic web site.

*Corresponding Author

The rest of this paper is organized as follows. Section 2 presents a brief summary of related research. Section 3 outlines our approach and presents our integer program. Section 4 presents our test data and results. Section 5 provides some conclusions and suggests future research.

2. Related Work

Strategies for sessionization can be classified as reactive and proactive [20]. Proactive sessionization strategies capture a rich collection of a user's activity during her visit to a site but are invasive and in some countries forbidden [20] or regulated by law to protect user privacy [16]. Examples include cookie oriented session retrieval [3], where personal data are stored on the user's computer and from which a complete session can be retrieved. URL rewriting [8] stores personal information that is finally stored on logs. The most invasive example is web tracking software, close to spyware, on a user's computer (or browser) that captures the entire session [17].

Reactive sessionization strategies have less privacy concerns because they only use web log register information, which does not include explicit user information [20]. However, a web log only provides an approximate way of retrieving a user's session for several reasons. The same IP address as recorded in the web log often contains the requests of several concurrent users without each user being uniquely identified [14]. Additionally, a user's activation of the back and forward browser button is often not recorded in the web log because, in most cases, the browser retrieves the page from its own cache. A proxy server, acting as an internet web page cache to reduce network traffic, can also capture web requests that are not recorded in a web log [11].

Prior methods to construct sessions from a web log have been heuristic and most commonly based on limited session duration [20]. These heuristics form one session at a time by grouping log registers from the same IP address and the same agent so that the session does not exceed a maximum duration parameter, usually 30 minutes [2]. After a session is formed, it is never changed even though its finalization can impact the construction of other sessions. Another heuristic approach is to construct sessions that share the same semantic [15].

Several authors have looked at the overall characteristics of sessions. They find that the size n of a web user session follows a power law $(n^{-\alpha}/\sum_k k^{-\alpha})$ distribution [12, 21]. The *size* of a session is the total number of registers in the session.

3. Our Approach

We propose sessionization using an integer program. Like the heuristic approaches it groups log registers from the same IP address and agent as well as ensuring the link structure of the site is followed in any constructed session. Unlike the heuristics, it does not construct the sessions one at a time but instead simultaneously constructs all sessions. The integer program is therefore able to determine, for example, the maximum number of sessions of a certain size whereas a heuristic can only provide a lower bound on this value. The ability of the integer program does come at the cost of increased solution time. The integer program we present here has been reformulated several times [7] to improve performance as we experimented with data. We find most instances of our integer program solve easily using the commercially available CPLEX solver [13]. For those not familiar with recent advances in linear (and integer) programming, Bixby [4], in a revealing article, compares different CPLEX versions and concludes in part *"a model that might have taken a year to solve 10 years ago can now solve in less than 30 seconds."* We report more on solution time in the section 4.

Each constructed session from a web log is an ordered list of log registers where each register can only be used once in only one session. Our integer program uses a binary variable X_{ros} that has value one if log register r is assigned as the *oth* request during session s and zero otherwise. Each index r identifies a unique register, each index s identifies a unique user session, and the index o is the ordered request of a register during a session. In the same session, it is possible for register $r2$ to occur immediately after register $r1$ if the two registers share the same IP address and agent, a link exists from the page requested by $r1$ to the page requested by $r2$, and the request time for register $r2$ is within an allowable time window since the request time for register $r1$.

We present the integer programming formulation below in NPS standard format [5].

3.1. Indices

o	Order of a log register visit during a session (*e.g.* $o = 1, 2, \cdots, 20$). The cardinality defines the maximum length of a session.
p, p'	Web page.
r, r'	Web log register.
s	Web user session.

3.2. Index Sets

$r' \in bpage_r$	The set of registers that can be the register immediately before register r in the same session. Based on: - pages that are available from the page of register r in one click - IP address matching of register r and register r' - agent matching of register r and register r' - time of register r and register r'. Of course, r can not occur before r' but we assume a user defined minimum and maximum time between two consecutive registers in the same session.

$r \in first$	set of registers that must be first in a session.

3.3. Data [units]

Used to produce the index sets above:

$time_r$	the time of register r [seconds].
ip_r	the IP address for register r.
$agent_r$	the agent for register r.
$page_r$	the page for register r.
$\underline{mtp}, \overline{mtp}$	the minimum, maximum time between pages in a session [seconds].
$adjacent_{p,p'}$	one if a page p' can be reached in one click from page p.

Used in formulation:

C_o	the objective function cost of having a register assigned to the *oth* position in a session.

3.4. Binary Variables

X_{ros}	1 if log register r is assigned as the *oth* request during session s and zero otherwise.

3.5. Formulation

Maximize $\sum_{ros} C_o X_{ros}$

Subject to:

$$\sum_{os} X_{ros} \leq 1 \qquad \forall r \qquad (1)$$

$$\sum_{r} X_{ros} \leq 1 \qquad \forall o, s \qquad (2)$$

$$X_{r,o+1,s} \leq \sum_{r' \in bpage_r} X_{r',o,s} \qquad \forall r, o, s \qquad (3)$$

$X_{ros} \in \{0,1\} \forall r, o, s,$
$X_{ros} = 0, \forall r \in first, o > 1, s$

The objective function expresses a total reward for sessions where a reward of $\sum_{o' \leq o} C_{o'}$ is obtained for any session of size o. As an example, setting $C_3 = 1$ and $C_o = 0 \quad \forall o \neq 3$ provides an objective function for maximizing the number of sessions of size three. Section 4 reports on how we varied the values of C_o and the results obtained.

Constraint set (1) ensures each register is used at most once. Constraint set (2) restricts each session to have at most one register assigned for each ordered request. Constraint set (3) ensures the proper ordering of registers in the same session. $X_{ros} \in \{0,1\} \forall r, o, s$ defines variables as binary. To improve solution time, we can fix (or eliminate) a subset of these binary variables to zero ($X_{ros} = 0, \forall r \in first, o > 1, s$). After forming the set $bpage_r$, the set $first$ is easily found ($r \in first$ if $bpage_r = \emptyset$).

3.6. Solving Instances

It is easy to construct instances of our integer program that can not be solved. For example, a web log of $100,000$ registers, such as the one we consider in our computational study, allowing a maximum of $5,000$ sessions, and a maximum session size of 20 produces 10^{10} binary variables and more than this number of constraints. Fortunately, a little processing helps to reduce an instance from a web log into separable instances or *chunks*.

We partition a web log into chunks where each one is formed such that no register in one chunk could ever be part of a session in another. This is easily accomplished by partitioning chunks so that each one corresponds to a unique IP and agent combination. Even after this partitioning, a chunk may contain too many registers to be solved easily. In such cases, a chunk may be further divided whenever the time difference between two consecutive registers (where the registers are sorted by time) exceeds $\overline{mtp}$.

Heuristic rules could be employed to continue to reduce the size of a chunk but such rules were not needed in our computational study. In fact, we avoided making a chunk too small because there is a fixed time associated with generating and solving each chunk. For our computational work, we used a minimum chunk size of 50.

4. Web Log Test Data

We consider a university web site (http://www.dii.uchile.cl) that hosts the main page of the Industrial Engineering Department of the University of Chile, sub-sites of research groups, personal homepages, a web mail site, academic programs and related project sub-sites. As a general purpose site, it has a lot of diversity and reasonably high traffic, although much of this traffic comes from web mail which we do not consider.

3, 756, 006 raw registers were collected over a time window of one month, April 2008. We want to find sessions consisting of just web pages so we filtered out multimedia objects (*e.g.*, pdf, jpg, and avi), faulty requests (HTTP errors), web spider requests, web mail requests, hacking attempts (very quick and continuous request from the same IP on some login pages), and monitoring tasks (bigbrother like systems). The final total for our study is 102, 303 clean registers of static html pages as well some dynamic pages (php and jsp), with a total of 172 different pages. Of these, 9, 044 registers correspond to visits to the root page of the site.

We find that only a few IP addresses account for the vast majority of all clean registers. Over 98 percent (16, 785 out of a total of 16,985 unique IP addresses) have less than 50 register for the entire month. Figure 1 displays the number of registers for the 100 IP addresses that account for the most registers.

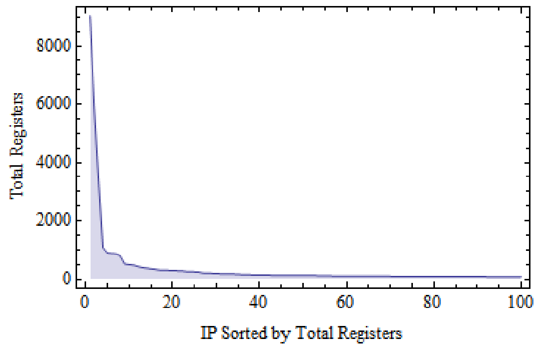

Figure 1. The number of registers for the 100 IP addresses that account for the most registers.

We also found how many unique web pages are visited by each IP address because we find IP addresses that visit many unique web pages have more diverse sessions. Figure 2 shows the number of unique pages requested by the 2,000 IP addresses that account for the greatest number of unique page requests. Of the IP addresses not shown, almost 84 percent (14, 265 out of 16, 985) visit three or less different pages for the entire month.

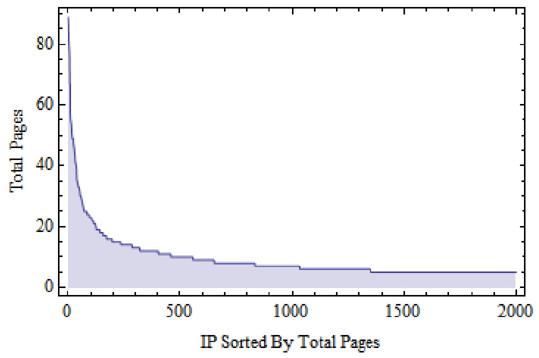

Figure 2. The 2,000 IP addresses that account for the greatest number of unique page requests.

We also performed a web crawling process on the site in order to recover its structure of hyperlinks, using the WebSphynx library [18]. We obtain 172 pages with 1, 228 links between them, for pages identified in the previous cleaned log registers. We store the information in a relational database (MySQL) that includes tables for unique IP addresses, unique page identifiers, and unique links between pages and the registers. The database maintains relational constraints between tables in order to ensure data consistency.

4.1. The Sessionization Experiments

We select the most relevant chunks for our sessionization study by IP address. For each IP address, we find the number of registers and a measure of the diversity of the pages visited over the registers. The measure of diversity we use is entropy, $S = \sum_p f_p Log_N(1/f_p)$, where f_p is the frequency of page p occurrence over all register entries for the same IP address and N is the number of unique pages visited by the same IP address. S takes values from zero to one. When the value of S is near zero, most register entries are for the same page, if the value is near one all the pages are visited with similar frequency. Figure 3 plots for each IP address the number of registers versus S. There are many IP addresses with diversity near zero (visiting one page most of the time) and many IP addresses with high diversity but a low number of registers for the entire month. We concentrated on the IP addresses with high diversity and a high number of registers reckoning that these are the most interesting (and most difficult to solve) for sessionization.

Selecting the IP addresses in the upper right rectangle of Figure 3 (more than 50 registers and S greater than 0.5) results in 130 IP addresses with 17,709 registers (17.3% of the total number of registers). We obtain 403 chunks when we partition these registers such that each chunk has at least 50 registers and $\overline{mtp} = 300$.

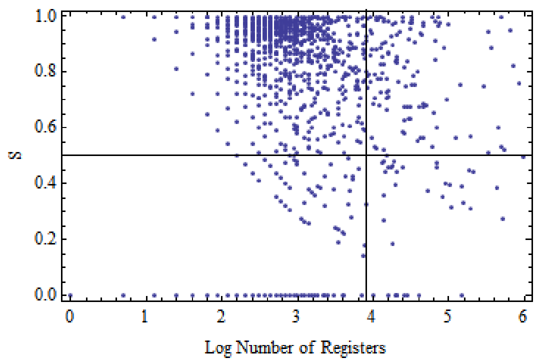

Figure 3. Log number of registers vs. S for each IP.

All computation is done using 1.6Ghz PC with two Gbs of RAM. We generate the integer program using GAMS [10] and solve it using CPLEX version 10.1.0 [9], controlled by a php script and MySQL 5.0.27 [1] as storage data engine.

The 403 different integer programs (with $\underline{mtp} = 0$, $\overline{mtp} = 300$, and a maximum session size of 20) range in size from about 4,000 to 292,000 binary variables and 8,000 to 281,000 constraints. For each integer program, we set the maximum time limit to 300 seconds, the *relative gap* to one percent, and the *absolute gap* to 0.9. With such limits, the CPLEX solver terminates when it has a solution guaranteed to be with one percent of optimal, or a solution guaranteed to have an objective function value within an absolute value of 0.9 from optimal, or it reaches a 300 second (five minute) limit. If it reaches a 300 second limit, it provides the best solution it has found by that time.

Without knowledge of the real sessions, it is not straightforward to measure the quality of sessions produced by a heuristic or our integer program. One measure that is available is how well the distribution of session sizes match the empirically observed power law distribution [12, 21]. We use linear regression on the logarithm of the size and the logarithm of the number of sessions. We report the regression correlation coefficient and standard error as our measures of sessionization quality. The closer the correlation coefficient is to one and the standard error near to zero, the better the sessionization result.

We performed many experiments with the objective function coefficients C_o and found most sets of values that reward sessions of longer size produced good quality sessions. Table 1 presents five sets of different objective function coefficient values used along with the resulting correlation coefficient and standard error. For example, we solved the integer program for all 403 chunks using the fifth set of coefficients, $C_o = o^2$ $\forall o$, and found the resulting correlation coefficient of 0.9607 and standard error of 0.5145. Figure 4 shows how many of each session size were found by our integer program for sizes two and higher, the power law distribution fit, and the resulting correlation coefficient for the third set of objective function coefficients.

	$C_o =$	R^2	StdError
1	$1/\sqrt{o}$	0.9222	1.0548
2	$Log(o)$	0.9752	0.4126
3	$3/2Log(o) + (o-3)^2/12o$	0.9784	0.3827
4	o	0.9742	0.4242
5	o^2	0.9607	0.5145

Table 1. Five sets of different objective function coefficient values and the resulting correlation coefficient and standard error.

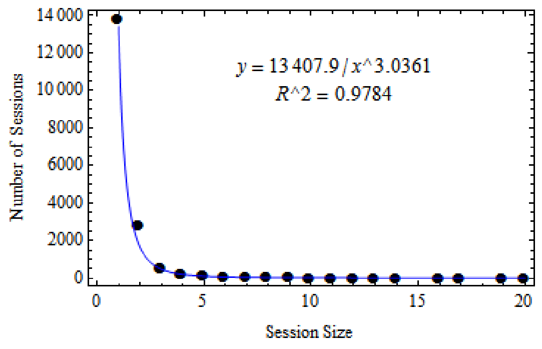

Figure 4. Session size found and the power law distribution fit.

The computation time was similar for all sets of coefficients. We report details for the third set. Over 85% (344 out of 403) of the chunks obtained a solution within one percent of optimal or within 0.9 of the optimal objective function value. The average relative gap for these 344 chunks was 1.1% indicating that almost all were terminated within one percent of optimal. The average generation and solution time for these chunks was only 11 seconds. The 300 second limit was reached in only 59 out of the 403 chunks. Figure 5 shows solution time by the number of binary variables. Above $230,000$ discrete variables we see most of the instances reach the $300s$ time limit.

For the 59 chunks reaching the limit, the average relative gap was 70%. The *relative gap* is the percentage difference between the best solution found and a theoretically best solution. We increased the solution time for several of these 59 chunks and found in most cases the additional time did not produce better solutions but did improve the bound on the theoretically best solution and therefore improved the relative gap. A further division of these 59 chunks into smaller chunks is another possible approach we tried on a few of these chunks but this too didn't produce better sessions.

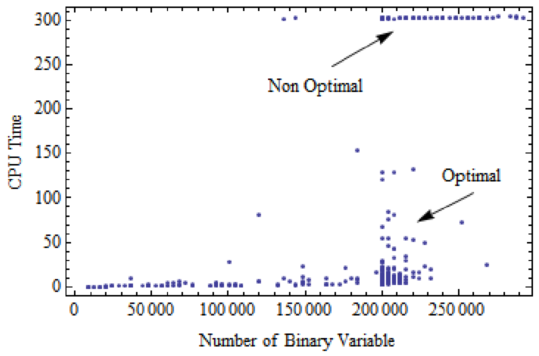

Figure 5. Solution time in seconds vs. number of binary variables.

For a variety of chunks, we also doubled the value of $\overline{mtp}$ to 600 seconds and the resulting sessions were almost identical to those found with $\overline{mtp} = 300$.

We also considered a change to the objective function so as to find the maximum number of sessions of a given size for use in other research [19]. Specifically, for a specific $size$ session, $C_o = 0 \ \forall o \neq size$, and $C_o = 1$ for $o = size$. Results for size two to six are shown in Table 2. We see that relatively few sessions of size six (or higher) are possible. These results simplify the task of finding combinations of parts of a session of fixed size.

Size	Num. Sessions
2	$3,000$
3	$1,509$
4	755
5	435
6	257

Table 2. The maximum number of sessions possible of a given size.

4.2. Comparing With a Time-Oriented Heuristic

We compare our results with a traditional sessionization timeout heuristic on all clean registers. The timeout heuristic is substantially faster (only 13 seconds) but results in a distribution of sessions with only a $R^2 = 0.9181$ correlation coefficient (not as good as the $R^2 = 0.9784$ found by the integer program) and a standard error of 0.6401 (nearly twice the standard error of 0.3817 found by the integer program). Figure 6 provides a comparison of the standard error for both the timeout heuristic and the integer program.

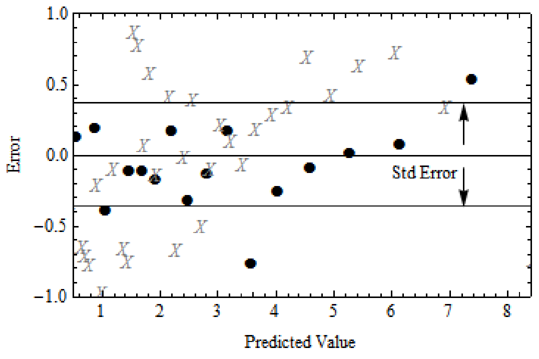

Figure 6. Predicted value error for the timeout heuristic shown as (X) and the integer program shown as (•). Also shown is the standard error (0.3817) band for the integer program.

5. Conclusions and Future Research

We present a new approach for sessionization using integer programming. When compared to a commonly used heuristic, we find the sessions produced by the integer program better match an expected empirical distribution. The integer program also allows us to determine the maximum number of sessions of a certain size.

Future research will focus on improving solution time. We will also change our integer program to seek the maximum number of a specific session(s) (pattern matching) that could be possible for a given log file. We will also enhance our integer program to model the possibility of a user's activation of the back and forward browser button.

Acknowledgement

This work has been partially supported by the National Doctoral Grant from Conicyt Chile and by the Chilean Millennium Scientific Institute of Complex Engineering Systems.

References

[1] C. Aulds. *High Performance MySQL.* O'Reilly Media, 2004.

[2] B. Berendt, A. Hotho, and G. Stumme. Data preparation for mining world wide web browsing patterns. *Journal of Knowlegde and Information Systems*, 1(1):5–32, 1999.

[3] B. Berendt, B. Mobasher, M. Spiliopoulou, and J. Wiltshire. Measuring the accuracy of sessionizers for web usage analysis. In *Proc.of the Workshop on Web Mining, First SIAM Internat.Conf. on Data Mining*, pages 7–14, 2001.

[4] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[5] G. G. Brown and R. F. Dell. Formulating integer linear programs: A rogues' gallery. *INFORMS TRANSACTIONS ON EDUCATION*, 7(2):1–13, 2007.

[6] R. Cooley, B. Mobasher, and J. Srivastava. Towards semantic web mining. In *Proc. in First Int. Semantic Web Conference*, pages 264–278, 2002.

[7] R. Dell, P. Román, and J. D. Velásquez. Identifying web user sessions using a discrete optimization approach. In *XIV Latin Ibero-American Congress on Operations Research (CLAIO)*, 2008.

[8] F. Facca and P. Lanzi. Recent developments in web usage mining research. In *DaWaK*, pages 140–150, 2003.

[9] GAMS Development Corporation. Gams/cplex, 2008. Solver CPLEX for GAMS (http://www.gams.com/dd/docs/solvers/cplex.pdf).

[10] GAMS Development Corporation. General algebraic modeling system (gams), 2008. Software available from http://www.gams.com.

[11] S. Glassman. A caching relay for the world wide web. In *Selected papers of the first conference on World-Wide Web*, pages 165–173, Amsterdam, The Netherlands, The Netherlands, 1994. Elsevier Science Publishers B. V.

[12] B. Huberman, P. Pirolli, J. Pitkow, and R. M. Lukose. Strong regularities in world wide web surfing. *Science*, 280(5360):95–97, 1998.

[13] ILOG. Ilog cplex, 2008. Software available from http://www.ilog.com.

[14] J.D.Velásquez and V. Palade. *Adaptive Web Sites: A Knowledge Extraction from Web Data Approach.* IOS Press, Amsterdam, NL, 2008.

[15] J. J. Jung and G.-S. Jo. Semantic outlier analysis for sessionizing web logs. In *ECML/PKDD Conference*, pages 13–25, 2004.

[16] D. Langford. *Internet Ethics.* MacMillan Press Ltd, 2000.

[17] V. Mayer-Schonberger. Nutzliches vergessen. In *Goodbye Privacy Grundrechte in der digitalen Welt (Ars Electronica)*, pages 253–265, 2008.

[18] R. C. Miller and K. Bharat. Sphinx: A framework for creating personal, site-specific web crawlers. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, pages 119–130, 1998.

[19] P. Román and J. D. Velásquez. Markov chain for modeling web user browsing behavior: Statistical inference. In *XIV Latin Ibero-American Congress on Operations Research (CLAIO)*, 2008.

[20] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *INFORMS Journal on Computing*, 15(2):171–190, 2003.

[21] A. Vazquez, J. G. Oliveira, Z. Dezso, K.-I. Goh, I. Kondor, and A.-L. Barabási. Modeling bursts and heavy tails in human dynamics. *PHYSICAL REVIEW E*, 73(3):036127, 2006.

[22] J. D. Zawodny. *Linux Apache Web Server Administration.* Sybex; 2 edition, 2002.