SWISH: Semantic Analysis of Window Titles and Switching History

Nuria Oliver, Greg Smith, Chintan Thakkar & Arun C. Surendran
Microsoft Research
Redmond, WA, USA
{nuria,gregsmi,acsuren}@microsoft.com
Chintan.S.Thakkar@iitkgp.ac.in

ABSTRACT

Information workers are often involved in multiple tasks and activities that they must perform in parallel or in rapid succession. In consequence, task management itself becomes yet another task that information workers need to perform in order to get the rest of their work done. Recognition of this problem has led to research on task management systems, which can help by allowing fast task switching, fast task resumption, and automatic task identification. In this paper we focus on the latter: we tackle the problem of automatically detecting the tasks that the user is involved in, by identifying which of the windows on the user's desktop are related to each other. The underlying assumption is that windows that belong to the same task share some common properties with one another that we can detect from data. We will refer to this problem as the *task assignment* problem.

To address this problem, we have built a prototype named SWISH that: (1) constantly monitors users' desktop activities using a stream of windows events; (2) logs and processes this raw event stream, and (3) implements two criteria of window "relatedness", namely the semantic similarity of their titles, and the temporal closeness in their access patterns.

In addition to describing the SWISH prototype in detail, we validate it with 4 hours of user data, obtaining task classification accuracies of about **70**%. We also discuss our plans on including SWISH in a number of intelligent user interfaces and future lines of research.

General Terms: Algorithms.
Keywords: Automatic task identification, user task modeling, clustering, user activity monitoring.

1. INTRODUCTION AND RELATED WORK

Bannon *et al.* [2] observed more than twenty years ago that information workers often switch between concurrent tasks or activities. This behavior has only increased in the years since, receiving increasing attention in the research community and the popular press. Numerous efforts have been made to assist information workers in this realm. We will refer to such multitasking assistance systems as "task management systems". Task management systems typically provide some efficient way of switching from one set of windows and applications to another set, as a basic form of task switching.

Shorly after Bannon's observations, Henderson and Card [13] noted that tasks could be supported via the management of "working sets" of windows, and identified desirable properties of these task management systems, including: fast task switching, fast task resumption, and easy reacquisition of the cognitive context associated with a task.

In addition to virtual desktop managers [13, 12], an array of novel solutions have been proposed, including extending the user's desktop with additional low-resolution screen space [3], using 3D cues as in the TaskGallery [23, 27], using zoomable interfaces as in Pad++ [4], and using time as a centralizing axis [21]. There also have been tiled window managers [7, 26], systems that bump irrelevant windows away [5, 17], systems employing a central focus region and a peripheral region for unused windows [22], and new, enhanced Taskbars [24]. Finally, there have been a number of systems that assist users in their task management tasks within a particular application, such as email [6].

In order to facilitate task management and task switching, these systems generally require knowledge of how the user's overall workspace is conceptually partitioned into the individual constituent tasks. MacIntyre [18] *et. al.* defined the concept of "working contexts" –coherent sets of tasks typically involving the use of multiple applications, documents and communication mechanisms with others. In their work, they recognized that a basic problem that needed to be addressed (but they didn't address due to its difficulty) was determining which documents are associated with each working context. We will name this problem the "task assignment" problem. Most systems rely on explicit user input for such knowledge, despite the extra overhead this imposes. There have been fewer efforts toward the automatic detection and recognition of the user's tasks from computer events, probably at least partially because of the difficulty of such an endeavor. For example, when a new document or window is opened, is it part of the current working context, the start of a new working context, or a signal to shift to some other existing working context? Finding answers to these questions is the main purpose and contribution of the SWISH system described in this paper.

The TaskTracer prototype by Dragunov *et al.* [25] ad-

dresses the task assignment problem, and is the most closely related work to ours in SWISH. TaskTracer is a desktop software system aimed at recording in detail how knowledge workers complete tasks, and intelligently leveraging that information to increase efficiency and productivity. The authors enumerate five desired capabilities of their system: more task-aware user interfaces, more efficient task-interruption recovery, better personal information management, workgroup information management and within-group workflow detection and analysis. TaskTracer tracks most interactions with a subset of typical desktop applications, including Microsoft's Office 2003, Microsoft's Visual Studio .NET, Windows XP and phonecalls. In their most recent work, the authors have added some supervised machine learning mechanisms for predicting the most likely folder (based on frequency of access) that the user would access next (FolderPredictor) and the most likely task that the user is engaged in (TaskPredictor).

We shall highlight now the main contributions of this paper as differences between the TaskPredictor component of TaskTracer and SWISH:

1. Instead of being restricted to a set of predefined applications, SWISH monitors events generated by any application on the Windows PC. Fenstermacher and Ginsburg [11] enumerated cross-application integration as one of the key features of any user activity monitoring software.

2. Unlike TaskPredictor, where users need to manually specify what tasks they are doing in the initial stage of data collection, we opted for a completely unsupervised approach. Users of SWISH do not need to label any of the tasks that they are performing at any instant of time.

3. Both TaskPredictor and SWISH include semantic analysis of window information. However, SWISH also incorporates temporal analysis of the patterns of window usage. We believe that it is important to leverage the complementarity of the information derived from analysis of different nature. We will return to this issue in Section 3.

4. SWISH records all application events in the background, in a completely unobtrusive manner, as the user works on the computer. Users do not need to interact with a special application as part of their work.

The rest of the paper is organized as follows. In Section 2 we describe in detail the architecture of the SWISH prototype. Section 3 is devoted to the description of the two algorithms that we implemented for clustering windows automatically. Experimental results to validate SWISH are in Section 4. Section 5 briefly presents our plans in incorporating SWISH in a number of user interfaces. Finally, Section 6 contains some conclusions and future lines of research.

2. ARCHITECTURE

The architecture of the SWISH prototype, depicted in Figure 1, consists of two complementary applications we have written to run on the Windows Operating System (OS), and an associated SQL Server relational database.

Our first application, LogFeeder, incorporates an adaptation of a logging tool called VibeLog [15] to provide the framework for computer activity collection. The Windows OS publicly exposes a collection of events keyed by the window handle (HWND) of the window on which the event occurred, and VibeLog programmatically hooks this collection (with the SetWinEventHook API) to receive events of interest and collect associated interesting properties of the relevant windows. These events are surfaced for every window on the system, regardless of the application that owns the window. The standalone version of VibeLog writes its time-stamped output sequentially to file in a form that can then be imported into a database for offline analysis. The purpose of LogFeeder is to take a stream of VibeLog output, either directly from the embedded version of a running VibeLog or from a previously recorded VibeLog session in the database, and replay it in real time as a "live" inter-process stream. Table 1 shows an abbreviated summary of the structure of a typical VibeLog output row.

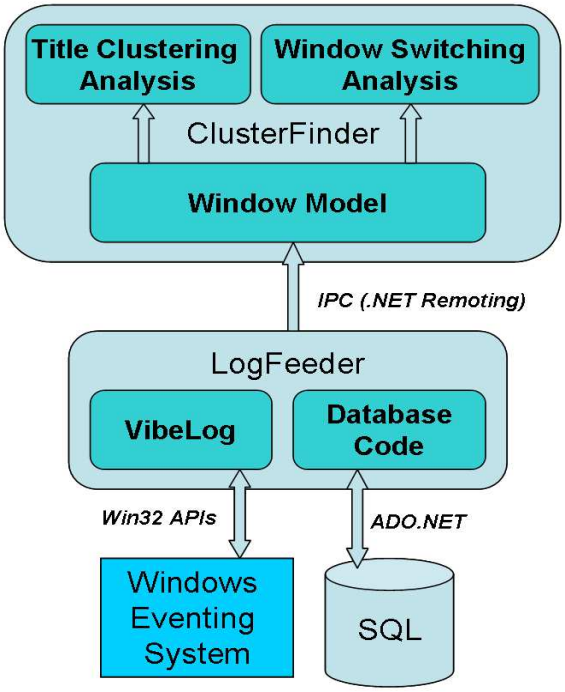


Figure 1: Block diagram of SWISH'es architecture.

Our second application, ClusterFinder, listens for this inter-process stream and uses it to build and maintain a live "window model" representing the window state of the system which generated the VibeLog output. This window model contains a list of objects representing the open windows on the system at any given moment in z-order (*i.e.*, "depth" within the overlapping window system), and tracks the history of the list and state changes to the windows in the list. We can then create various modular algorithms that analyze window-based user activity solely by querying ClusterFinder's window model. The advantage of this infrastructure is that it allows us to experiment with analyzing the live local session, as well as to record specific session streams and iteratively evaluate the efficacy of our

Table 1: Description of the VibeLog output stream

Name of event in stream	Description	Example value
Event number	Monotonically increasing event row number	548
Elapsed ms	Millisecond timestamp	703656
Event name	NAME CHANGE, CREATE, ACTIVATE, *etc.*	SHOW
HWND	Win32 window handle	$0x324E5$
Title	Current title of window	"Re:Review"
ClassName	Current window class name	Tooltip_32
Process Name	Name of executable which owns the window	Outlook.exe
Position	Current window coordinates	10,4,73,14
StyleBits	A bit field representing the window's "style" and "extended style" attributes. (WS_VISIBLE, WS_CHILD, *etc.*)	$0x96000000$ $0x96000000$ $0x00000088$

algorithms by tweaking and testing them repeatedly against the exact same session stream.

Basing our design on the VibeLog stream dictates that the input features of our analysis algorithms are necessarily restricted to what can be found in the VibeLog stream, and the usefulness of the analysis is potentially dependent on the "correctness" of the window model being reconstituted by the ClusterFinder application. This is complicated by the fact that the original Windows OS event stream is not itself 100% reliable –it sometimes omits events or contains event sequences that violate the conceptual window state machine (*e.g.*, two DESTROY events for the same window handle). We are continually working on improving the robustness of our window model in the face of a limited, noisy event stream, and we are encouraged by the results so far. As suggested by the screenshot in Fig 2, even without any representation of window contents the graphical reconstruction of the session appears to be quite evocative of the actual user experience.

3. AUTOMATIC WINDOW CLUSTERING

The data analysis in the SWISH prototype is derived from the assumption that windows that are related to each other –based on a particular criterion– belong to the same task. This assumption is motivated by the observation that tasks typically involve multiple windows and those windows share some features. We focus in this paper on two sets of features:

1. **Semantic features**: Windows associated with the same task share common words in their content, and, in particular, in their window titles.

2. **Temporal features**: Windows associated with the same task are accessed in temporal proximity to each other, *i.e.*, input focus switches between windows belonging to the same task occur more frequently than switches to windows outside the task.

Next, we shall describe in detail each of these two types of analysis.

3.1 Clustering Based on Window Titles

The title of any arbitrary window on a Windows PC is easily accessible through public Win32 programming interfaces (APIs), regardless of the application that the window belongs to. The application is responsible for putting text in this title that represents something meaningful about the window's content to the user. In this part of the analysis we were interested in leveraging this lightweight, readily accessible information. Specifically, we were interested in exploring whether we could reliably and successfully identify windows that are related to each other by semantically analyzing their titles. With this goal in mind, we reviewed a spectrum of popular information retrieval techniques, focussing on unsupervised document clustering approaches.

3.1.1 Data Representation

Any text processing algorithm requires a convenient representation of the corpus. Vector space models offer a *term x document* matrix representation of the corpus in which each document is a vector with terms being the dimensions. In the simplest kind of representation, the ij^{th} entry of the matrix represents how many times the i^{th} term appeared in the j^{th} document. The entry might be further processed to be some function of this value as described in Section 3.1.3.

Clustering is possible because of the assumption that underlying the corpus of data there is a small set of *concepts* that these documents are about. Clustering algorithms usually resort to dimensionality reduction techniques which map documents from the high dimensional term space to the low dimensional concept space. In the reduced term space, we would like the semantically similar documents to be nearer while those that are dissimilar to be apart. This reinforces the idea that there is a latent structure in the corpus. In an ideal case, we would want to faithfully reconstruct the generative model that generated the corpus. A generative model gives a precise methodology for constructing documents once the parameters are specified.

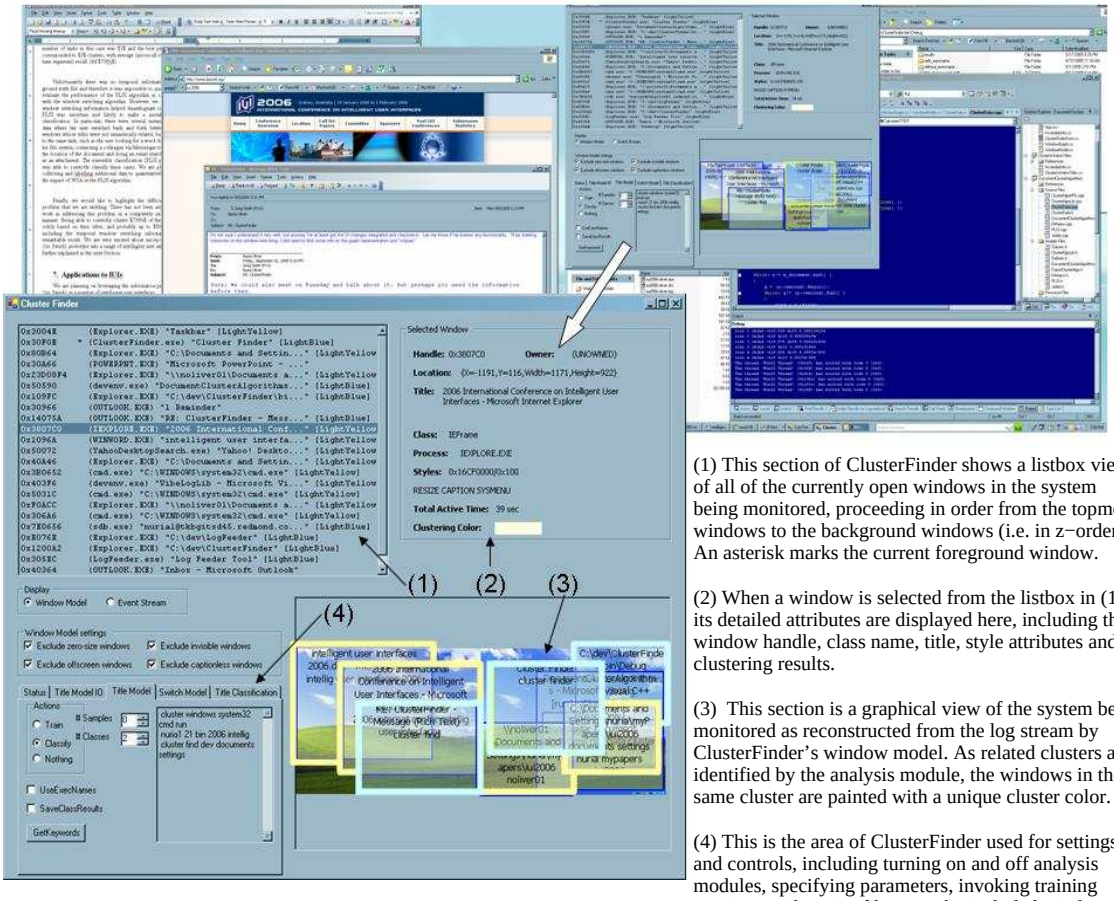
In SWISH we opted for statistical generative approaches. In particular and after empirical experiments comparing the performance various algorithms, we determined that the Probabilistic Latent Semantic Indexing (PLSI) algorithm [14] was the best suited to our problem. We shall describe this algorithm in some detail below.

3.1.2 Title Preprocessing

Typical word preprocessing in information retrieval includes ignoring words that have no relationship to a window's content, such as articles, conjuctions, prepositions, *etc.* (so-called "stop-words"), and generalizing the remaining words by stemming them[1]. In our system, the fact that our source words are limited to those contained in the window title text presents some additional challenges compared to full-document modeling. Some of the peculiarities of window titles include:

1. **Short in length:** Window titles are typically just a few words long. Due to this scarcity of data, it is

[1]In SWISH we use a variation of the stemmer by Porter [20].



(1) This section of ClusterFinder shows a listbox view of all of the currently open windows in the system being monitored, proceeding in order from the topmost windows to the background windows (i.e. in z−order). An asterisk marks the current foreground window.

(2) When a window is selected from the listbox in (1), its detailed attributes are displayed here, including the window handle, class name, title, style attributes and clustering results.

(3) This section is a graphical view of the system being monitored as reconstructed from the log stream by ClusterFinder's window model. As related clusters are identified by the analysis module, the windows in the same cluster are painted with a unique cluster color.

(4) This is the area of ClusterFinder used for settings and controls, including turning on and off analysis modules, specifying parameters, invoking training sessions, and setting filters on the included windows.

Figure 2: Screenshot of a typical user's desktop running the SWISH prototype.

critical to ensure that the algorithms model only the information-carrying words.

2. **Additional stop-words:** Most window titles contain application-dependent words that have no relationship to the content of the window. For example, "Microsoft Internet Explorer" appears in every in Internet Explorer window title. In this phase of the processing, we eliminate all application-dependent keywords.

3. **Arbitrary words:** It is not unusual for titles to have arbitrary words that are the result of composing multiple words into one long word. For example, a document could be named "hawaii_vacation_summer_2005.doc", and this document name might be put verbatim into the window title by the application. In an ideal case, we would like to split such a long word into its smaller, meaningful units, *i.e.*, "hawaii", "vacation", "summer", "2005". SWISH contains a module for splitting long words (*i.e.*, words whose length is above a threshold) into its smaller components.

This preprocessing of the window titles has a significant impact in the overall performance of the system, as described in Section 4. At the end of this step, we have "clean" versions of the titles, ready to undergo the next step in the analysis: feature extraction.

3.1.3 *Feature Extraction*

The most basic representation of the document corpus consists of the raw frequencies of occurrence of terms in documents, *tf*. This representation has the disadvantage that commonly occurring terms may unnecessarily make all documents look similar even when they are not characteristic of a particular document. To compensate for the problem ellicited above, the inverse document frequency measure or *tfidf* adds a weight to the raw frequencies, corresponding to the inverse frequency of documents *idf*, $tfidf = tf * idf$. The *idf* measure for a term i is given by $idf_i = \log \frac{|D|}{(|D_i|+1)}$, where $|D|$ is the total number of documents and $|D_i|$ is the number of documents containing term i. Intuitively, *idf* scales down commonly occurring terms and scales up words which rarely occur in documents and therefore are probably distinctive for that document. The *tfidf* measure can be viewed as the mutual information between terms and documents [1].

In SWISH we obtained best results using the *tfidf* measure and that is the measure used in the experiments reported in Section 4.

3.1.4 *Model Learning: Probabilistic Latent Semantic Indexing (PLSI)*

Probabilistic Latent Semantic Indexing [14] defines a generative model for the document-term pair (d_i, t_j), with $i = 1, ..., N$ and $j = 1, ..., M$. It assumes that every term-

document pair is independent given a hidden topic z, $z = 1, ..., K$. The graphical model for PLSI is depicted in Figure 3. From the Figure, the probability of (d_i, t_j), $P(d_i, t_j)$ is given by $P(d_i, t_j) = P(d_i) \sum_{zZ} P(t_j|z_k)P(z_k|d_i)$, where $P(t_j|z_k)$ and $P(z_k|d_i)$ form the model parameters that need to be estimated from data. Note that, once the parameters are learned, PLSI does not provide a mechanism to assign these values to an unseen document whose terms are all new to the model. In that sense, PLSI would not be a truly generative model, but an explanatory model for the corpus. The parameters can be efficiently estimated using the EM algorithm [10] such that the likelihood of the corpus is maximized. The M-step equations are given by:

$$P(t_j|z_k) = \frac{\sum_{i=1}^{N} n(d_i, t_j)P(z_k|d_i, t_j)}{\sum_{j=1}^{M}\sum_{i=1}^{N} n(d_i, t_j)P(z_k|d_i, t_j)} \quad (1)$$

$$P(z_k|d_i) = \frac{\sum_{i=1}^{M} n(d_i, t_j)P(z_k|d_i, t_j)}{n(d_i)} \quad (2)$$

where M is the number of terms or words, N the number of documents and Z the number of concepts.

The E-step equation is given by:

$$P(z_k|d_i, t_j) = \frac{P(t_j|z_k)P(z_k|d_i)}{\sum_{k=1}^{K} P(t_j|z_k)P(z_k|d_i)} \quad (3)$$

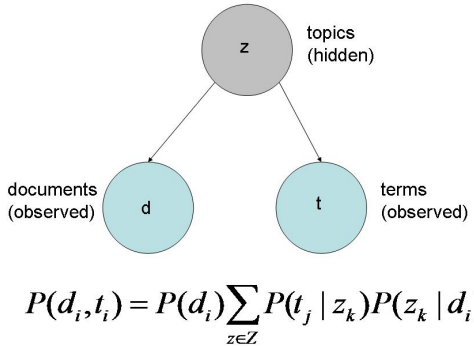


Figure 3: Graphical model representing the PLSI model. Note how documents (d) and terms (t) are independent from each other given the hidden topic (z).

To get a clustering from the model parameters, we just need to look at $P(z_k|d_i)$ for all k and assign document i to the cluster that maximizes that probability. Note that it is assumed that the number of hidden topics, Z, is known *a priori*.

As with any local optimization algorithm, the EM algorithm used to estimate the parameters of the PLSI model is sensitive to its initialization point. In order to make sure that the algorithm starts at a reasonable initial point, we use the K-means algorithm to obtain the initial values of the parameters. In the experiments described in Section 4 we learned 30 models with 30 different initializations and computed their average performance.

Note that PLSI is a soft-clustering technique (*i.e.* probabilistic). It has been reported in the literature to perform better than other clustering techniques like agglomerative clustering or CEM (Classification EM)[19]. In our experience, PLSI also had the best performance when compared to other clustering algorithms such as CEM or K-means.

Figure 4 illustrates the title clustering module explained above.

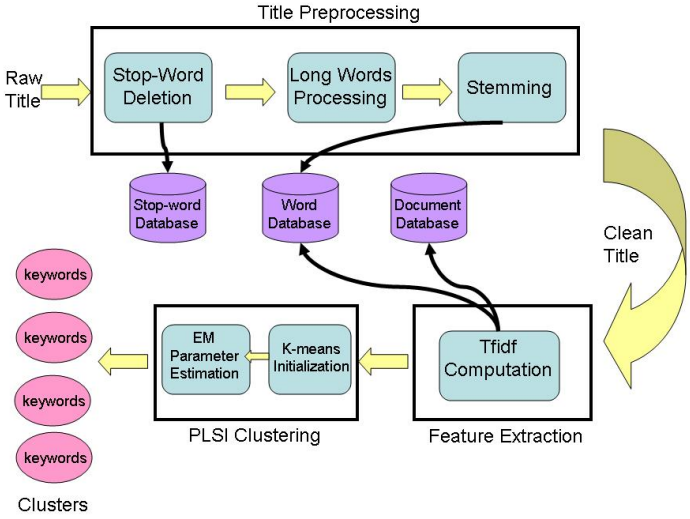


Figure 4: Block diagram of the title clustering module.

3.2 Clustering Based on History of Window Switches

The second module of analysis in SWISH is inspired by the observation that related windows are typically used in temporal sequence. SWISH keeps track of all the window switching events that have taken place during a time interval of length T, and automatically builds a *window switching* matrix, WS, where each element ws_{ij} is proportional to the number of times that the user switched from window w_i to window w_j during the time period T.

We shall describe next the "window switching analysis" or WSA, as it is illustrated in Figure 5. The *window switching* matrix is used to build a directed graph (top, left graph in Figure 5), where each node represents a window, the edges in the graph correspond to transitions between windows and their weight is proportional to the number of switches from the window where the edge originates (parent) to the window where it ends (child). This graph is then moralized [16] and pruned (top, right graph in Figure 5), eliminating edges with weight less than a certain threshold[2], corresponding to spurious transitions. Finally, the graph's maximal cliques are automatically identified via the Bron-Kerbosch algorithm [8] (bottom, right graph in Figure 5). Each of the cliques contains the windows that are related to each other, and therefore are assumed to belong to the same task. This is depicted in the bottom, left graph of Figure 5.

Note that, due to the moralization of the graph, not all the nodes (windows) in a clique need to have had transitions among themselves.

In the experiments described in Section 4 we used a time window T of length 5 minutes.

[2]In the experiments described in Section 4, we use 3 for the threshold.

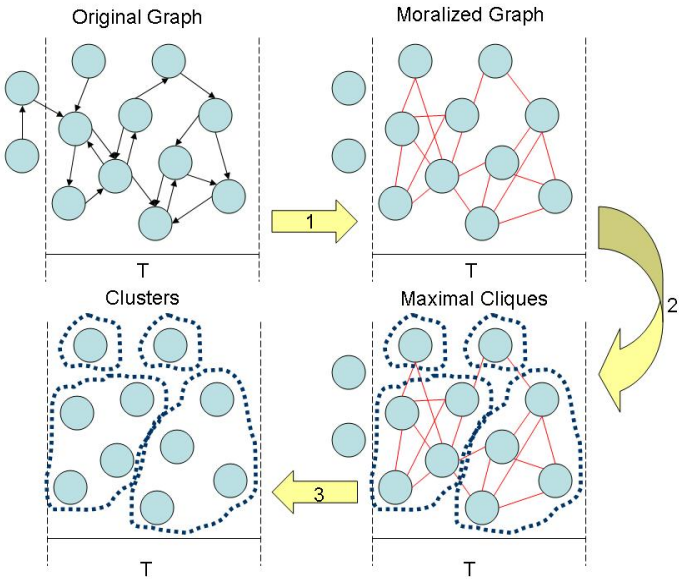


Figure 5: Data flow of the window switching algorithm.

3.3 Ensemble Classification

An interesting and open research problem is that of combining the outputs of different classifiers into one unified answer. In the current version of the SWISH prototype, we have included a first approach to ensemble classification. First, SWISH clusters the windows based on their title information via the PLSI algorithm. It then only resorts to the window switching algorithm when the semantic title processing module is not sufficiently (*i.e.*, above a given threshold) confident about where a particular window belongs. This situation typically arises when (1) there is a new title with unseen words by the system, or (2) the probability that the title belongs to each of the clusters is below a certain threshold, *i.e.*, the title does not fit well in any of the clusters.

We are currently exploring additional algorithms for ensemble classification, including using spectral clustering techniques from information retrieval [9] to cluster the window titles, and incorporating the window switching history directly in the graph that the spectral clustering techniques build. With this approach we would have the same representation and mathematical framework in both the semantic and temporal analysis.

4. EXPERIMENTAL RESULTS

4.1 Experimental Design

Evaluating unsupervised learning algorithms is not an easy task. One standard way of doing it is by first manually labeling the data to create the ground truth class assignments –usually a very tedious, labor-intensive and expensive process. Then the clusters are formed via the algorithm that needs evaluation. Finally, cluster identification is carried out, which is usually done by looking at the % of documents in a cluster that belong to a class. The largest % is assumed to be the label of the class. Two quantitative measures are typically extracted: *precision*, *i.e.*, what % of documents in a cluster belong to the class assigned to that cluster, and *recall*, *i.e.*, what % of all documents that belong to that class appear in this cluster. The $F1$ measure balances both recall and precision in a single number:

$$F1 = \frac{2 * precision * recall}{precision + recall} \quad (4)$$

where the higher the value of the $F1$ measure, the higher the performance of the algorithm.

4.2 Results

To quantitatively evaluate the SWISH prototype we collected and manually labeled over 4 hours of user data for a single user. There were 5 different main tasks that the user worked on during that period of time. However, not all windows in the system during that period of time belonged to one of those tasks. In particular, about $20 - 30\%$ of the windows were spurious windows that didn't belong to any of the main tasks. Therefore *recall* is a more meaningful measure than *precision* in this context, because the clusters will contain a significant number of windows that are noise in the system, *i.e.*, do not really belong to any of the clusters.

In a first set of experiments, we compared the precision and recall of SWISH with and without preprocessing the window titles, as described in Section 3.1.2. The impact of preprocessing the titles was highly significant, as reflected on Table 2. In this case, the optimal number of clusters was 5 and the measures are the averages over 30 runs.

Table 2: Average precision and recall measures with and without preprocessing the window titles (5 clusters).

	With	Without
Precision	**0.49**	0.39
Recall	**0.72**	0.65

Figure 6 contains the average recall, precision and F1 measure of the PLSI algorithm over 30 runs and with an increasing number of clusters. Note how the optimal number of clusters is 5, which corresponds to the true number of tasks.

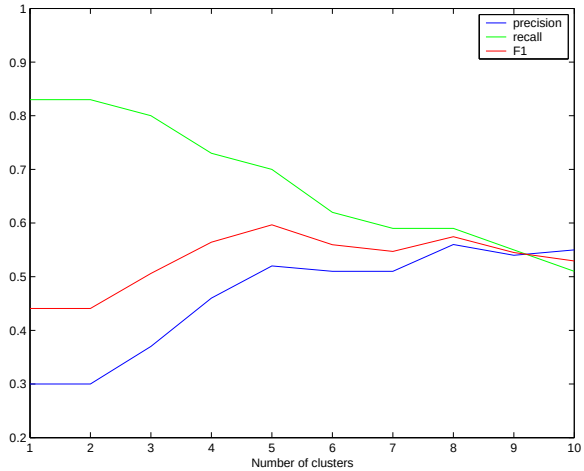


Figure 6: Average precision, recall and F1 measure for 2 to 10 clusters.

We ran a second set of experiments where we divided the data into 1-hour chunks and we clustered those smaller sets. SWISH performed better clustering these smaller sets of data than the entire corpus. Intuitively, this makes sense: the user worked on fewer, more coherent tasks during the 1-hour periods than during the entire 4-hour session. The average number of tasks in this case was 3 and the best performance corresponded to 3 clusters, with average (across all runs and all time segments) recall **76**%.

To have a better idea of the contents of the clusters, Figure 7 illustrates a few exemplary titles and the top 3 keywords of the results of clustering a 30 minute portion of the data[3]. In this case, the optimal number of clusters was 3. Note how the titles belong to very different applications. For example, in cluster number 2, the first two titles are from a Microsoft Word document, the third title is from Microsoft PowerPoint, and the fourth and fifth from Internet Messenger.

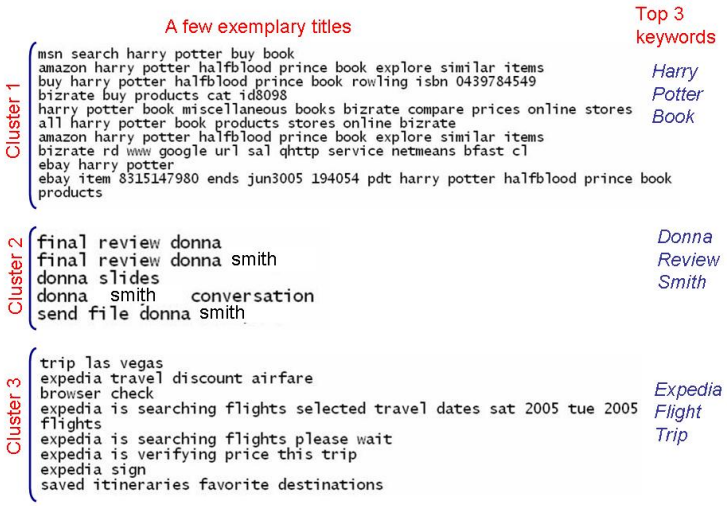


Figure 7: Exemplary titles and top 3 keywords when clustering 30 minutes of user data with 3 clusters.

Even though we only had ground truth data for one user, we ran SWISH on over 30 hours of data of six additional subjects. Our anecdotal experience with these data mirrors the quantitative results provided above.

Unfortunately there was no temporal information in the ground truth file and therefore it was impossible to quantitatively evaluate the performance of the PLSI algorithm in conjunction with the window switching algorithm. However, we found that window switching information helped disambiguate cases where PLSI was uncertain and likely to make a mistake in the classification. In particular, there were several instances in the data where the user switched back and forth between a few windows whose titles were not semantically related, but belonged to the same task, such as the user looking for a word document in his file system, contacting a colleague via Messenger to ask about the location of the document and doing an email search to find it as an attachment. The ensemble classification (PLSI plus WSA) was able to correctly classify these cases. We are planning on collecting and labelling additional data to quantitatively evaluate the impact of WSA in the PLSI algorithm.

Finally, we would like to highlight the difficulty of the problem that we are tackling. There has not been any previous work in addressing this problem in a completely unsupervised manner. Being able to correctly cluster over 70% of the windows, solely based on their titles, and probably up to 80% when including the temporal window switching information is a remarkable result. We are very excited about incorporating the SWISH prototype into a range of intelligent user interfaces, as further explained in the next Section.

[3]We have changed the last name appearing in the titles to preserve privacy.

5. APPLICATIONS TO IUI

We are planning on leveraging the information provided by SWISH in a number of intelligent user interfaces.

First, we are working on extending the Groupbar prototype [24] to automatically: (1) propose groups of windows to the users, and (2) assign labels to the clusters, based on the keywords for each cluster. These two new extensions of the Groupbar were actually high in the list of suggested improvements proposed by users as a result of a user study. Moreover, both pieces of information are already available in SWISH.

Second, we are working on an implicit query system that will display relevant information to the window that the user is currently engaged with. We plan on using the keywords of the cluster that the window belongs to as the words to query on.

Third, we are implementing an automatic "desktop cleanup" application that will automatically propose to close unused, unrelated windows, *i.e.*, windows that do not belong to any of the current tasks.

Finally, we are incorporating SWISH's automatic window clustering to a task management prototype that will enable users to efficiently and seamlessly create, switch to, handle and resume tasks.

6. SUMMARY AND FUTURE WORK

The work of information workers today is increasingly fragmented by multitasking. Tasks usually involve documents and processes of a heterogeneous nature. With increased multitasking, support for task management is itself becoming a critically important task in supporting the rest of the information worker's work. In this paper we address the challenge of task assignment, *i.e.*, figuring out which task the user is involved in at each instant of time, and which documents or windows are part of that task. We believe that this constitues an important step towards more efficient and automatic task management systems.

In particular, we have developed a prototype named SWISH for automatically detecting groups of windows that are related to each other. We assume that windows belonging to the same task share certain features. In SWISH we analyze windows according to two types of features: (1) semantic clustering of windows based on their titles, and (2) temporal clustering of windows based on how they are accessed by the user. We have presented results in over 4 hours of real data, obtaining task classification accuracies of about 70%, *i.e.*, SWISH assigned windows to the correct task 70% of the time. We find our results highly encouraging and we believe they constitute a significant step towards solving the task-assignment problem in a completely unsupervised, unobtrusive and automatic manner.

As previously described, we are currently working on incorporating SWISH in a number of intelligent user interfaces. In addition, we are exploring strategies for "ensemble classification", *i.e.*, combining the outputs of the different window classifiers in a mathematically sound way. We are also very interested in combining long-term with short-term user models, automatically estimating the optimal number of tasks at any instant of time, and dynamically updating the models once they become obsolete. Finally, we are looking into other types of features that could be incorporated into our ensemble classification to help improve task assignment results.

7. ACKNOWLEDGEMENTS

The authors would like to thank John Winn and his team for sharing their graphical models code, and Tara Matthews for her valuable comments on the paper.

8. REFERENCES

[1] Aizawa. The feature quantity: An information theoretic perspective of tfidf-like measures. *Information Processing and Management*, 39:1:pp. 45–65, 2003.

[2] L. Bannon, A. Cypher, S. Greenspan, and M.L. Monty. Evaluation and analysis of users' activity organization. In *Proc. SIGCHI conf. on Human Factors in Computing Systems (CHI'83)*, pages pp. 54–57, 1983.

[3] P. Baudisch, N. Good, and P. Stewart. Focus plus context screens: combining display technology with visualization techniques. In *Proc. of UIST'01*, pages pp. 31–40, 2001.

[4] B. Bederson and J. Hollan. Pad++: A zooming graphical interface for exploring alternative interface physics. In *Proc. ACM symposium on User interface software and technology (UIST'94)*, pages pp. 17–26, 1994.

[5] B. Bell and S. Feiner. Dynamic space management for user interfaces. In *Proc. ACM symposium on User interface software and technology (UIST'00)*, pages pp. 238–248, 2000.

[6] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *Proc. SIGCHI conf. on Human Factors in Computing Systems (CHI'03)*, pages pp. 345–352, 2003.

[7] S. Bly and J. Rosenberg. A comparison of tiled and overlapping windows. In *Proc. SIGCHI conf. on Human Factors in Computing Systems (CHI'86)*, pages pp. 101–106, 1986.

[8] C. Bron and J. Kerbosch. Algorithm 457 –finding all the cliques of an undirected graph. *Communications of the ACM*, 16(9):pp. 575–577, 1973.

[9] S.C. Deerwester, S. Dumais, T.K. Landauer, G.W. Fumas, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):pp. 391–407, 1990.

[10] A.P. Dempster, N.M. Laird, and D.B Rubin. Maximum likelihood from incomplete data via de *em* algorithm. *Journal of the Royal Statistical Society*, 39-B:pp. 1–38, 1977.

[11] K.D. Fenstermacher and M. Ginsburg. A lightweight framework for cross-application user monitoring. *Computer*, 35(3):pp. 51–59, 2002.

[12] A. Goldberg. *Smalltalk-80*. Addison-Wesley, 1983.

[13] A. Henderson and S. Card. Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics (TOG)*, 5(3):pp. 211–243, 1986.

[14] T. Hofmann. Probabilistic latent semantic indexing. In *Research and Development in Information Retrieval*, pages pp. 50–57, 1999.

[15] D.R. Hutchings, G. Smith, B. Meyers, M. Czerwinski, and G. Robertson. Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *Proc. Conf. on Advanced Visual Interfaces (AVI'04)*, pages pp. 32–39, 2004.

[16] F.V. Jensen. *Bayesian Networks and Decision Graphs*. Springer Verlag, 2001.

[17] E. Kandogan and B. Schneiderman. Elastic windows: evaluation of multi-window operations. In *Proc. SIGCHI conf. on Human Factors in Computing Systems (CHI'97)*, pages pp. 250–257, 1997.

[18] B. MacIntyre, E.D. Mynatt, S. Voida, K.M. Hansen, J. Tullio, and G.M. Corso. Support for multitasking and background awareness using interactive peripheral displays. In *Proc. ACM symposium on User interface software and technology (UIST'01)*, pages pp. 34–43, 2003.

[19] M. Meila and D. Heckerman. An experimental comparison of several clustering and initialization methods. *Machine Learning*, 42:9–29, 2001.

[20] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):pp. 130–137, 1980.

[21] J. Rekimoto. Time-machine computing: A time-centric approach for the information environment. In *Proc. ACM symposium on User interface software and technology (UIST'99)*, pages pp. 45–54, 1999.

[22] G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D.R. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable fabric: flexible task management. In *Proc. Conf. on Advanced Visual Interfaces (AVI'04)*, pages pp. 85–89, 2004.

[23] G. Robertson, M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The task gallery: a 3d window manager. In *Proc. SIGCHI conf. on Human Factors in Computing Systems (CHI'00)*, pages pp. 494–501, 2000.

[24] G. Smith, P. Baudisch, G. Robertson, M. Czewinski, B. Meyers, D. Robbins, and D. Andrews. Groupbar: The taskbar evolved. In *Proc. OZCHI'03*, pages pp. 41–50, 2003.

[25] S. Stumpf, X. Bao, A. Dragunov, T.G. Dietterich, J.L. Herlocker, K. Johnsrude, L. Li, and J. Shen. Predicting user tasks: I know what you're doing! In *National Conference on Artificial Intelligence (AAAI'05)*, pages pp. –, 2005.

[26] W. Teitelman. *Methodology of Window Management*, chapter Ten years of window system - A retrospective view. Berlin: Springer-Verlag, 1986.

[27] H. Wurnig. Design of a collaborative multi-user desktop system for augmented reality. In *Proc. Central European Seminar in Computer Graphics*, 1998.