MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY
http://www.merl.com

Automatic Structuring of Embedded Hypermedia Documents

Rebecca Hwa — Harvard Univ.
Joe Marks — MERL
Stuart Shieber — Harvard Univ.

TR-95-6 February 1995

Abstract

Embedded hypermedia documents (HDs) are becoming more common in aircraft cockpits, power- and industrial-plant control panels, and other user interfaces to complex systems. These hypermedia documents are often large and heavily cross referenced, yet they must support extremely efficient and intuitive user navigation. Designing such documents well is difficult. In this paper, we describe a computer-based approach to structuring HDs. We show how an interface designer can quantify ease of navigation in a way that makes the HD-structuring problem equivalent to a version of the well-known optimization problem of graph partitioning. This reduction to graph partitioning is the basis for an implemented system that uses known graph-partitioning heuristics to structure HDs automatically.

Keywords: display ergonomics, information display aids, human-machine interaction.

1. First printing, TR95-6, February 1995. Revised April 1996.

1 Introduction

The bridge of a supertanker, an aircraft cockpit, a control panel for a nuclear-power plant: these are among the most complex and sophisticated user interfaces ever designed. Nowadays, such interfaces typically contain one or more embedded hypermedia documents (HDs). They are like most other HDs in that each page of the document contains information in the form of text and graphics, and navigational links whereby the user can move to other pages. However, the performance requirements for these HDs are far greater than for most hypermedia applications.

Facilitating efficient and intuitive navigation is the central design problem for HDs embedded in complex-system interfaces. In this paper we describe a simple set of metrics for stating and quantifying ease of navigation. When stated in terms of these metrics, HD structuring reduces to a variant of graph partitioning, a well-known combinatorial optimization problem.[1] Existing heuristics for graph partitioning can then be applied to yield a near-optimal document structure.

The method has been applied to the design of one of the embedded HDs in the cockpit of the U. S. Army's Comanche helicopter. For expository purposes, we illustrate the approach in this paper by considering the design of a simple HD for a hypothetical bank ATM; ATM interfaces are probably the most familiar examples of embedded HDs. In the next section, we describe our technical approach in detail with the aid of simple worked ATM examples. We then describe extensions to our basic approach, and conclude with a discussion of future work.

2 Technical Approach

2.1 Problem statement

A *display item* (DI) comprises the symbols or text that are displayed as an atomic unit to provide information or allow for an action. In the ATM example, these include DIs for withdrawing money or checking an account balance. The complete list of DIs for the ATM example is given in Table 1. For each DI we also give the area required for its display, and the symbol by which we will refer to it throughout the rest of the paper.

To generate ease-of-navigation metrics for a HD, we must give the interface designer an ability to state and quantify information about how the display items will be accessed and perceived. We provide two simple predicates for this purpose, SEQUENCE and CLUSTER. The SEQUENCE predicate is used to indicate the order in which DIs are accessed by a user in the performance of a typical task. The CLUSTER predicate is used to identify DIs that share a common property, one important enough to have them located near each other in the HD. Each predicate instance can be assigned an *importance weight* by the designer. This weight might be based on something as concrete as usage-pattern data, or something as vague as

[1] The general utility of mathematical graphs for modeling aspects of HD structuring has been noted previously by others [7].

Action	*Area*	*DI Symbol*
Welcome message	2.0	`WELCOME`
Checking balance	1.5	`X_BAL`
Savings balance	1.5	`SAV_BAL`
Deposit	2.0	`DEP`
Withdrawal	2.0	`WD`
Checking-to-savings transfer	1.5	`X_SAV_TRNSFR`
Savings-to-checking transfer	1.5	`SAV_X_TRNSFR`
$20 fast-cash withdrawal	1.5	`$20`
$50 fast-cash withdrawal	1.5	`$50`
$100 fast-cash withdrawal	1.5	`$100`
Exit dialogue	2.0	`EXIT`

Table 1: DI data for a hypothetical ATM example.

the designer's intuition. A sample set of design predicates for the ATM example is given in Table 2.

In addition to a set of design predicates, we need two more data to complete a problem instance: the area of the display screen[2] and the area required for navigational links.[3] For our hypothetical example, we assume a screen area of 7.0, and a navigation-link area of 1.0.

The statement of our example problem is now complete. The computer's task is to assign DIs to pages and to instantiate navigational links where needed so that the following ease-of-navigation metrics are minimized: (i) the number of inter-page moves required to traverse each DI sequence, weighted by the importance of the sequence; and (ii) the distribution of conceptually clustered DIs across multiple pages, weighted by the importance of the cluster. In the next two subsections we show how this computational task can be solved via a reduction to the problem of graph partitioning, and by the subsequent application of a known graph-partitioning heuristic.

2.2 Reduction to graph partitioning

Given a graph $G = (V, E)$, where V is a set of vertices and E is a set of weighted edges that connect the vertices, the traditional graph-partitioning problem is to divide V into k equal-sized subsets $\{V_1, \ldots, V_k\}$ such that the sum of the weighted edges connecting vertices in different subsets (the size of the *cut set*) is minimized. We will use a slightly different variant of graph partitioning in which vertices can have different weights, and the equal-cardinality requirement for each partition is replaced by the requirement that the sum of the vertex

[2]We make the simplifying assumption that a set of DIs can be simultaneously displayed on a screen if their areas do not sum to more than the screen area. This is of course not true in general. We describe later how this assumption might be relaxed.

[3]Like most embedded HDs, ATM interfaces use "soft" buttons whose functionalities change from screen to screen. The current mapping of the soft buttons must therefore be indicated on the screen. The area required to indicate the current mapping is what we denote as the navigational-link area.

SEQUENCE *Predicate*	*Weight*
WELCOME →X_BAL →WD →EXIT	15
WELCOME →X_BAL →SAV_BAL →X_SAV_TRNSFR →EXIT	10
WELCOME →DEP →X_BAL →EXIT	15
WELCOME →$20 →EXIT	15
WELCOME →$50 →EXIT	15
WELCOME →$100 →EXIT	15
WELCOME →SAV_BAL →SAV_X_TRNSFR →EXIT	15

CLUSTER *Predicate*	*Weight*
{$20, $50, $100}	100
{WELCOME, EXIT}	100

Table 2: Design predicates for the hypothetical ATM example.

weights in any partition must not exceed some specified amount.

We reduce HD structuring to graph partitioning by mapping DIs to graph vertices, and SEQUENCE and CLUSTER predicates to graph edges. The precise nature of the mapping relationship is best illustrated by example: in Figures 1–6 we create the graph-partitioning instance that corresponds to the ATM example described in Table 1.

We begin by creating a graph vertex for each DI listed in Table 1, as shown in Figure 1. The weight of each graph vertex is set equal to the area required for its corresponding DI. Then we add edges that correspond to the design predicates from Table 2. The first SEQUENCE predicate causes three new edges to be added, which connect the vertices as shown in Figure 2. The weight of each new edge is the importance weight of the SEQUENCE predicate that engendered the edge, 15 in this case. The second SEQUENCE predicate causes only three new edges to be created, because one already exists between the vertices corresponding to the DIs WELCOME and X_BAL. The new edges are given a weight of 10, and the weight of the existing edge (WELCOME, X_BAL) is increased to 25 (see Figure 3). The graph after all SEQUENCE predicates have been processed is shown in Figure 4.

Next, we consider the CLUSTER predicates in Table 2. They are reflected in the graph by making the subgraphs consisting of vertices {$20, $50, $100} and {WELCOME, EXIT} into completely connected subgraphs by introducing or modifying edges as before (see Figure 5).

If we were now to partition the graph in Figure 5, we would necessarily obtain a solution with at least one DI on every page: this is intrinsic to the concept of graph partitioning. However, it is not always desirable. For example, many well-structured HDs have pages that contain no DIs, only navigational-link buttons. A more subtle limitation of a naive partitioning approach is that it cannot allow for *indirect linkage* (i.e., when the user moves from one page to another via a third page that contains no DI of interest) to follow a DI sequence. Fortunately, we can overcome this limitation by an additional final step in the reduction: for each edge (A, B), we create a new *"stepping-stone" vertex* $v_{\mathtt{AB}}$ of size 0, and we

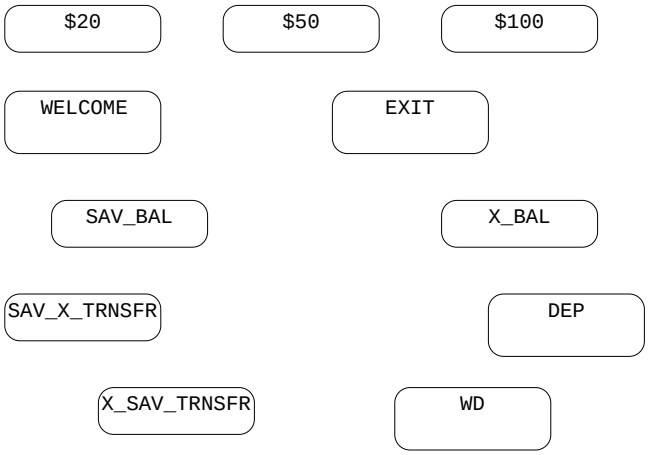

Figure 1: Graph vertices for the hypothetical ATM example.

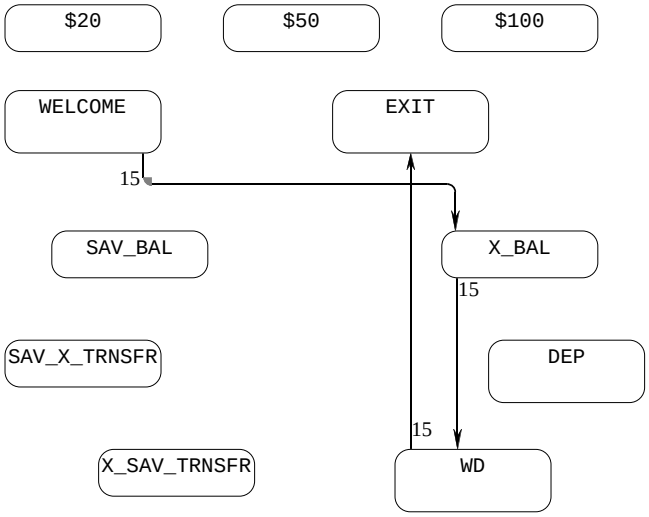

Figure 2: The graph after processing the first SEQUENCE predicate.

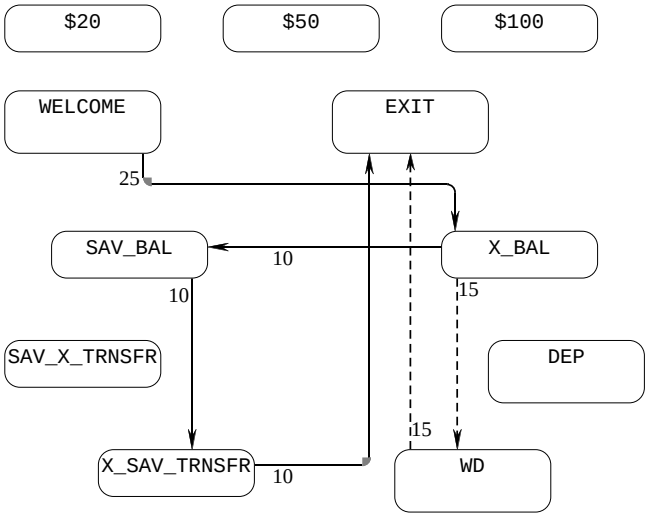

Figure 3: The graph after processing the second SEQUENCE predicate. New or modified edges are drawn with a solid line, unmodified edges with a dotted line.

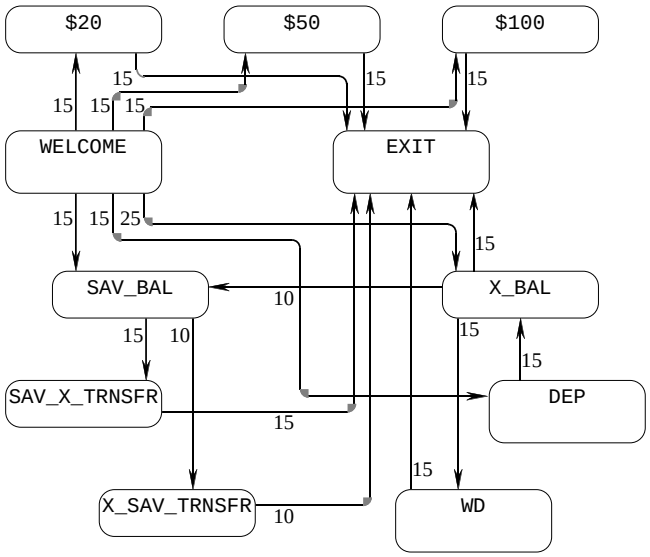

Figure 4: The graph after all SEQUENCE predicates have been processed.

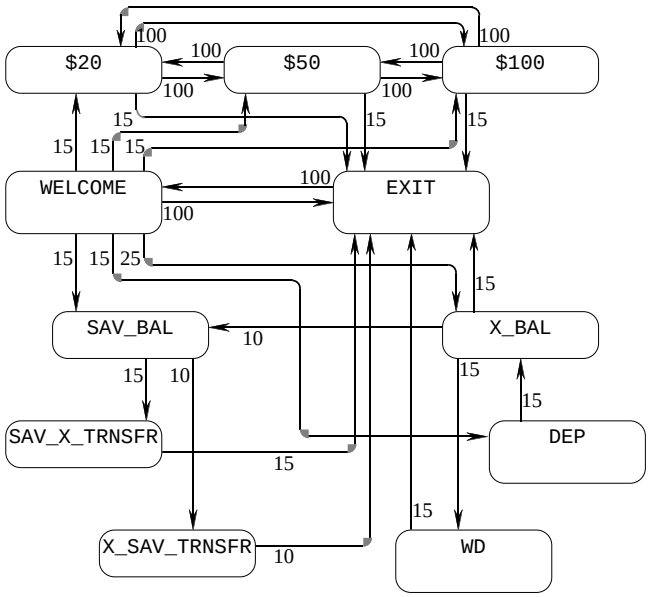

Figure 5: The graph after processing the CLUSTER predicates.

replace edge $(\mathtt{A}, \mathtt{B})$ with edges $(\mathtt{A}, v_{\mathtt{AB}})$ and $(v_{\mathtt{AB}}, \mathtt{B})$; the new edges have the same weight as the edge they replace. The final graph is shown in Figure 6. In the next subsection we will see how a minimum-cost partitioning of this graph will correspond to a document structure that is optimal with regard to the ease-of-navigation metrics described at the end of the problem statement.

2.3 Partitioning the graph

Once the complete graph has been generated, a good partitioning of it must be found. Although graph partitioning is NP-complete [2], near-optimal solutions can usually be obtained using heuristic methods. The Kernighan-Lin (KL) algorithm is a standard method for graph bisection, the special case of graph partitioning that arises when the vertex set is to be split into only two partitions [3].[4] The basic KL algorithm can be readily generalized to account for: more than two partitions; weighted vertices; weighted edges; and various constraints on the sizes of the vertex subsets, such as requiring that no subset exceed a certain size. We use a variant of the KL algorithm that is generalized appropriately for our application: a simple preprocess is used to generate an initial partitioning that satisfies the area constraint;[5] if two vertices in the graph are connected by an edge and are assigned to different subsets in the partitioning, then a navigational link is instantiated to connect the two pages corresponding to the subsets; the area taken up on a page is computed by summing the areas required for

[4]Recent research has produced a host of new graph-partitioning algorithms, some of which are demonstrably superior to the KL algorithm [1]. We used the KL algorithm because it is simple and relatively easy to modify and generalize.

[5]The KL algorithm requires an initial valid partitioning as input.

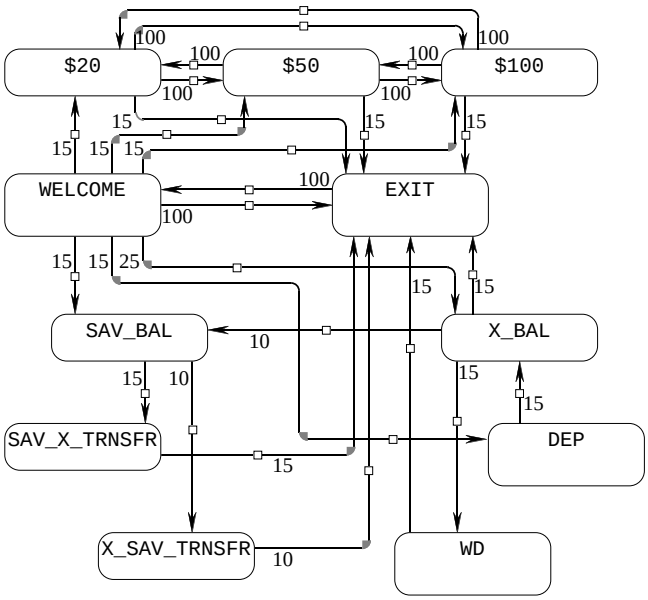

Figure 6: The complete graph after the processing of all design predicates and the introduction of stepping-stone vertices.

its DIs and navigational links.

We ran the algorithm 100 times on the graph in Figure 6, with a different initial partitioning each time. (Each run took approximately 10 seconds on an HP 9000/735 workstation.) Three of these runs produced the same partitioning, one with a cut-set size of 220 (see Figure 7), which results in the document structure summarized in tabular form in Table 3 and depicted graphically in Figure 8. The document structure found for this simple example resembles the ones usually used in actual ATMs: the initial page presents the welcome message and contains a menu of all available transactions; the other pages contain the DIs necessary for these transactions, along with a link back to the initial page. The utility of stepping-stone vertices is demonstrated by the indirect linkage involved in following the second DI sequence in Table 2: After visiting the `X_BAL` DI on the Checking page, the sequence is completed by moving to the Savings page via the Start page.

Given the design predicates in Table 2, the structure in Figure 8 is probably optimal. However, a different set of reasonable design predicates can lead to a very different document structure. For instance, starting from the alternative design predicates in Table 4, 22 of 100 runs of the partitioning algorithm generated the solution illustrated in Table 5 and Figures 9 and 10. The document structure is distinguished by its exclusive use of forward links: the user can only move forwards in the document, but not backwards. Although this structure may seem strange, it is a very good solution given the design specification in Table 4.

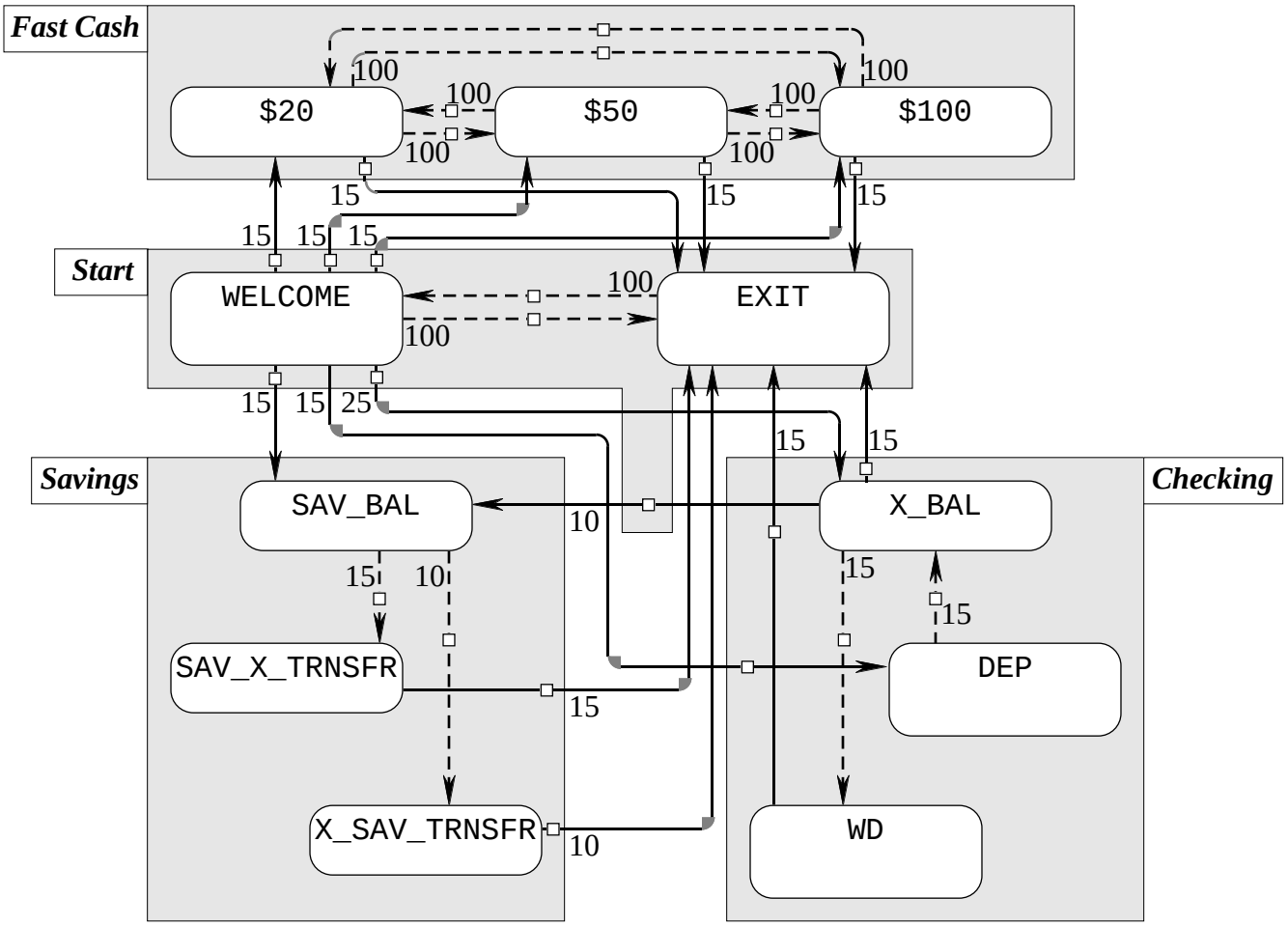

Figure 7: The partitioned graph. The rectangular boxes indicate the vertex partitions, each tagged with a mnemonic partition name. Graph edges internal to a partition, shown as dotted lines, do not contribute to the cut-set cost, which is the sum of the inter-partition edge weights.

Page	*DIs Assigned*	*Links*	*Area*
Start	`WELCOME` `EXIT`	Fast Cash Checking Savings	7.0

Fast Cash	`$20`
	`$50`
	`$100`
		Start	5.5

Checking	`X_BAL`
	`DEP`
	`WD`
		Start	6.5

Savings	`SAV_BAL`
	`X_SAV_TRNSFR`
	`SAV_X_TRNSFR`
		Start	5.5

Table 3: The document structure engendered by the partitioned graph.

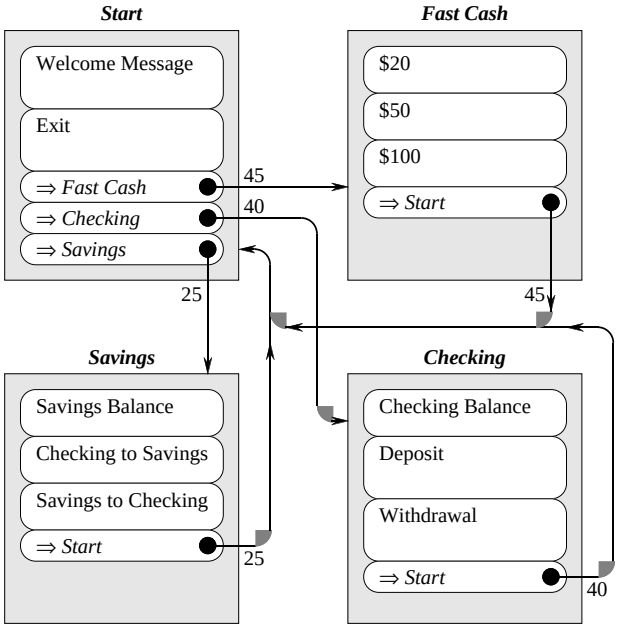

Figure 8: The document structure engendered by the partitioned graph. Numbers on the links are the summed weights of edges between the appropriate partitions. Note that the areas of the items on each page, both the DIs and links, fit within the area limit.

SEQUENCE *Predicate*	*Weight*
WELCOME →DEP →WD →EXIT	10
WELCOME →X_BAL →WD →SAV_X_TRNSFR →EXIT	5
WELCOME →SAV_BAL →SAV_X_TRNSFR →EXIT	5
WELCOME →DEP →X_SAV_TRNSFR →SAV_BAL →EXIT	5
WELCOME →X_BAL →WD →EXIT	5
WELCOME →X_BAL →$20 →EXIT	20
WELCOME →X_BAL →$50 →EXIT	20
WELCOME →X_BAL →$100 →EXIT	30

CLUSTER *Predicate*	*Weight*
{$20, $50, $100}	100

Table 4: Alternative design predicates for the hypothetical ATM example.

Page	*DIs Assigned*	*Links*	*Area*
Start	WELCOME X_BAL	Checking Savings Fast Cash Exit	6.5
Checking	DEP WD	Savings Fast Cash/ Exit	6.0
Savings	SAV_BAL SAV_X_TRNSFR X_SAV_TRNSFR	Fast Cash/ Exit	5.5
Fast Cash/ Exit	$20 $50 $100 EXIT		6.5

Table 5: The document structure engendered by the new partitioning.

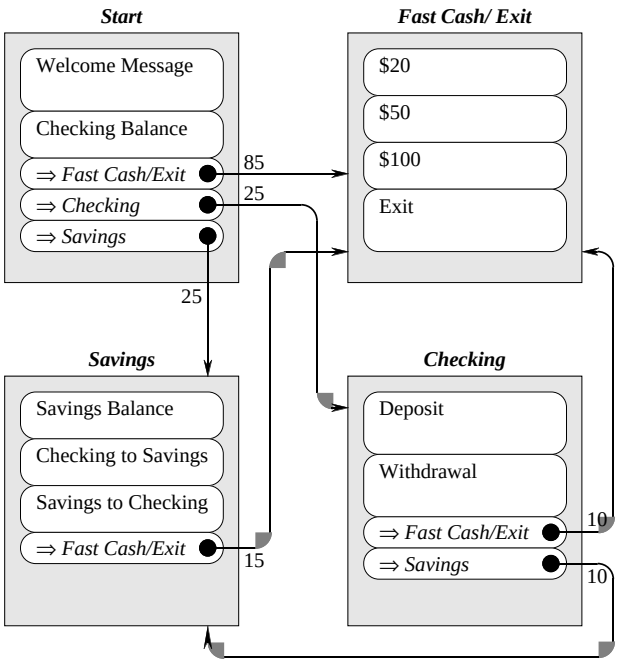

Figure 9: The document structure engendered by the new partitioning.

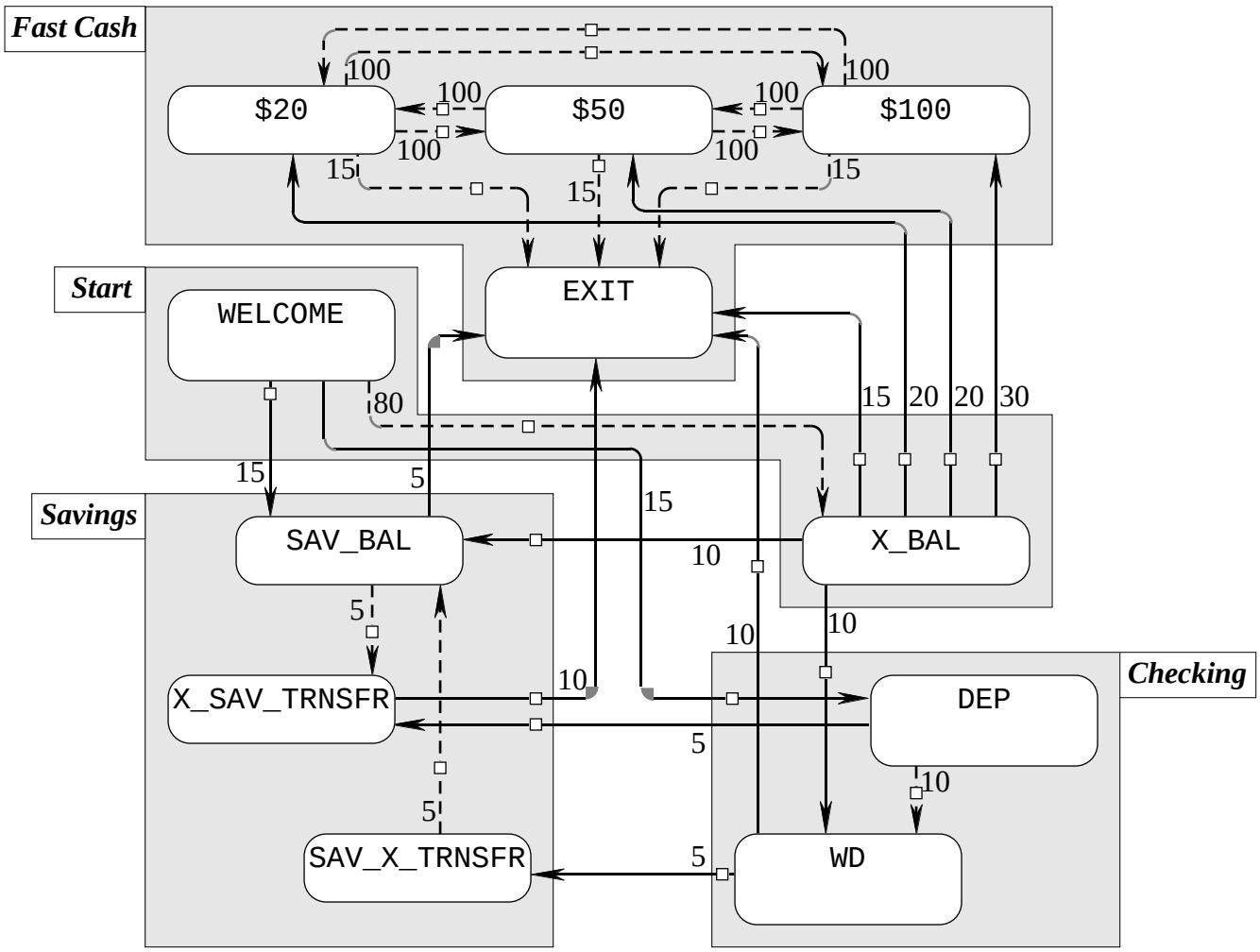

Figure 10: A different partitioning: cut-set size = 140.

3 Extensions

The simple examples discussed in the previous section do not illustrate some important issues that arise in more realistic examples. In this section, we examine the most significant issues in more detail, and describe how they might be addressed in our framework.

3.1 Enriching the Input Specification

DI sequences and clusters are not the only design predicates with which the interface designer might like to express design desiderata. One of the advantages of our approach is the ease with which other design considerations can be incorporated, and the designer's vocabulary commensurately increased. For example, the inverse of the cluster predicate, the ANTICLUSTER predicate, can be used to identify pairs of DIs that are conceptually distinct.

For example, to avoid confusion one might want to discourage close association between the DIs that relate to checking and savings accounts. This distinction would be reflected in the document structure by placing each DI in an anticlustered pair on a different page. Anticlusters can be incorporated into our approach by adding edges with negative weights to a graph during the reduction step.

Another potentially useful design predicate is MAXIMALLY ACCESSIBLE. Maximally accessible DIs are those that should be quickly accessible from anywhere in the document. In the ATM example, `EXIT` is a good example of a DI that one might want to make maximally accessible. This design predicate can be incorporated into our approach by adding positively weighted edges between each graph vertex and the one that corresponds to the maximally accessible DI.

3.2 Other Extensions

The two refinements described in the previous subsection are straightforward, but several other desirable extensions will be more difficult to achieve:

- We have assumed for simplicity that the area required for a DI can be represented satisfactorily as a one-dimensional value. While this is the case for some hypermedia documents (e.g., typical HTML documents), in general it is not true: even if the sum of the display areas for a set of DIs is less than the page size, there is no guarantee that the DIs can be placed on the page without overlap, let alone be placed in an aesthetic or logical arrangement. Incorporating page layout into our approach might be accomplished using existing techniques [4, 5, 6].

- Suitable page titles are essential for HD navigation: each navigational link must indicate the title of the page to which it points. However, sometimes the most efficiently navigable documents will have pages that contain disparate DIs, making it very difficult to title those pages appropriately. It may be possible to utilize hierarchic taxonomies of domain-dependent terms to generate page titles automatically, or to generate ANTICLUSTER predicates that will help prevent unfortunate groupings of DIs on the same page.

- We envisage any automatic HD-structuring system being used in an iterative fashion, with the designer making repeated changes to the design predicates, and the computer generating many different HD structures. It is therefore desirable that a small change to a set of design predicates lead to a small change in document structure. Unfortunately, this is not true of our current system: small changes in the design predicates can result in large changes to the resulting document structure. Incorporating some stability into the graph-partitioning process may require the development of new graph-partitioning algorithms.

- In some instances the placement of duplicate DIs on different pages can result in easier navigation. For example, including the `EXIT` DI on more than one page in our example ATM documents would allow the user to exit the document more conveniently. However, allowing for duplicate DIs is not readily accommodated in the current reduction to graph partitioning. Solving this problem may also require the development of novel partitioning algorithms.

4 Conclusions and Future Work

By reducing the HD-structuring problem to graph partitioning, we can make use of heuristic algorithms to generate HD structures automatically. The method described here has been tested on one of the embedded HDs from the cockpit of the U.S. Army's Comanche helicopter. This HD contains approximately 70 pages. Our system produced reasonable structures for it within a few hours on a workstation. However, as we have indicated above, our experience on this project led us to conclude that several extensions will be necessary before our system is practically useful for real-world applications. Nevertheless, we are optimistic that these extensions can be realized with future work.

In the future, we hope to experiment with other application domains using the same technique. For instance, telephone-menu systems exhibit some of the characteristics of HDs (with space limitations in visual HDs being replaced by time limitations in the aural regime), and could potentially benefit from better structuring. More speculatively, it may be possible to use similar techniques to restructure existing HDs based not on a designer's specification of access data, but rather on profiling the usage of the document itself.

Acknowledgments

We would like to acknowledge the collaboration of the following people on the formulation of the Comanche-helicopter HD: Dave Davis, Betsy Constantine, and Jim Kelly of TICA Associates, Cambridge, Massachusetts; and Barry Smith and his colleagues at NASA Ames, California. We also thank Wheeler Ruml for helpful discussions about graph partitioning. Hwa and Shieber were funded in part by NASA through a subcontract from TICA Associates and by National Science Foundation Grant IRI-9350192.

References

[1] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: The VLSI Journal*, 19:1–81, 1995.

[2] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[3] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

[4] Won Chul Kim and James D. Foley. Providing high-level control and expert assistance in the user interface presentation system. In *Proceedings of INTERCHI '93*, pages 430–447, Amsterdam, The Netherlands, April 1993.

[5] Andrew Sears. AIDE: A step toward metric-based interface development tools. In *Proceedings of the Eighth Annual Symposium on User Interface Software and Technology (UIST '95)*, pages 101–110, Pittsburgh, PA, November 1995.

[6] Louis Weitzman and Kent Wittenburg. Automatic presentation of multimedia documents using relational grammars. In *Proceedings of the Second Annual ACM Conference on Multimedia*, pages 443–451, San Francisco, CA, October 1994.

[7] Yan Yufik and Thomas Sheridan, 1995. Personal communication.