Adaptive Mobile Interfaces Through Grammar Induction

Jun Kong
North Dakota State University
jun.kong@ndsu.edu

Keven L. Ates Kang Zhang
University of Texas at Dallas
{atescomp, kzhang}@utdallas.edu

Yan Gu
North Dakota State University
yan.gu@ndsu.edu

Abstract

This paper presents a grammar-induction based approach to partitioning a Web page into several small pages while each small page fits not only spatially but also logically for mobile browsing. Our approach proceeds in three steps: (1) using the grammar induction technique to generate a graph grammar, which formalizes design policies for presenting information in a clear and logic structure; (2) based on the graph grammar, a graph parser parses a Web page to recover the hierarchical logic structure underlying that Web page; (3) the extracted logic structure models the content organization in the Web page, and is used to partition the Web page into several small pages for mobile displays.

1. Introduction

Wireless network and mobile devices make it possible to access information from anywhere at anytime. The ubiquitous access to Web information, however, raises a new challenge to the Web page presentation. Currently, most Web pages are designed for personal computers. Without adaptation, these pages on the small screen of mobile devices require users to frequently scroll the display window to find the content of interest, which makes mobile browsing frustrating and annoying.

Markup languages, such as WML and XHTML Mobile Profile (XHTML-MP), have been proposed to support the rendering on mobile devices. Though the number of Web pages in the form of WML or XHTML is growing fast, Web pages tailored to personal computers still dominate the Internet. Furthermore, keeping two versions of presentations, one for desktops and the other for mobile devices, is time-consuming and error-prone in maintenance. Therefore, it is desirable to automatically adapt a Web page from a desktop presentation to a mobile presentation whenever needed.

One challenging issue in the adaptive presentation is to discover closely related information in the original Web page, i.e. page segmentation. We can classify the approaches of page segmentation into two categories: website-independent heuristic approaches and website-dependent formal approaches.

The heuristic approaches [6, 7, 12, 17, 18] analyze either the HTML elements or the visual layout of a Web page to detect closely related contents. Those approaches assume that Web designers follow some common guidelines to present information. However, different designers can have different understandings on the same guideline, which can cause an inconsistency in the presentation across different Web sites. The inconsistency can reduce the accuracy of page segmentation. Furthermore, those heuristic approaches only provide coarse page segmentations which cannot recognize the semantic relation between different blocks.

Zhang *et al.* [10, 21] proposed a formal approach for page segmentation based on graph grammars. Instead of requiring a common pattern of design and presentation across different Web sites, this approach only assumes a consistent pattern applied to Web pages within a Web site. Based on this assumption, design patterns are formalized as a graph grammar, which specifies the information organization in a Web page. The graph grammar can be designed to derive a fine-grained segmentation, such as recognizing the title, publishing time and content in an article. Based on the graph grammar, a graph parser parses a Web page and produces a hierarchical structure, which indicates the content organization. However, it is time-consuming to design a graph grammar, which limits its application in practice.

In summary, the heuristic approaches suffer from inconsistent designs and presentations and only provide a coarse-grained segmentation, while the grammar-based formal approaches are not scalable to different Web sites. This paper presents a novel approach, which combines the grammar-based approach with grammar induction. From a Web page, the grammar induction can automatically derive a graph grammar, which formalizes the underlying information organization. In other words, the grammar induction process can replace human experts in designing a graph grammar used to guide the page segmentation process.

The rest of the paper is organized as the following. Section 2 gives the overview of our approach. Section 3 introduces the basic concepts of graph grammar and grammar induction. Section 4 illustrates how to construct and refine a graph grammar through grammar induction.

Section 5 goes through the process of page segmentation and gives an optimized parser. Section 6 discusses an adaptive layout. Section 7 compares related work, followed by the conclusion in Section 8.

2. Overview

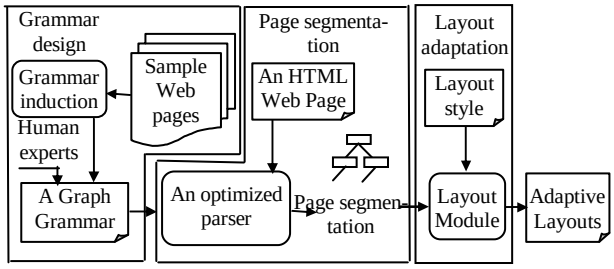


Figure 1: Overview of an adaptive presentation

Figure 1 gives an overview of our approach, which mainly includes three parts: grammar design, page segmentation and layout adaptation. In order to keep a consistent layout across a Web site, Web designers in general follow some guidelines to organize and present information. Such guidelines can be formalized as a graph grammar. Instead of designing the graph grammar from scratch, the grammar induction process automatically generates from sample Web pages a graph grammar, which is verified and elaborated by human experts. This automation significantly reduces the effort of designing a graph grammar. Based on the graph grammar, a graph parser parses a Web page to derive the hierarchical organization of contents underlying a Web page. Finally, according to a specific layout policy, such as adapting the original Web page to a one-column presentation, the layout module produces an adaptive layout suitable for mobile display and browsing.

Based on the derived graph grammar, a graph grammar parser is used to segment a Web page. Therefore, the time complexity of the grammar parser is critical to the overall performance. However, even for the most restricted classes of graph grammars, the membership problem is NP-hard [15]. The high time complexity is mainly caused by backtracking during the parsing process. We design an optimized grammar parser, which automatically applies grammatical rules in a certain order based on the hierarchical nature of information organization. The optimized parser avoids backtracking and runs in polynomial time.

3. Preliminary

3.1. Spatial graph grammar

A graph grammar includes a set of grammatical rules (i.e. productions). Each production has two graphs, called *left graph* and *right graph*, and models a local information organization. For example, a museum is displaying its collections online. The presentation of each collection includes a picture and a textual description, enclosed in a table. Such an information organization can be formalized as a production in Figure 2(a).

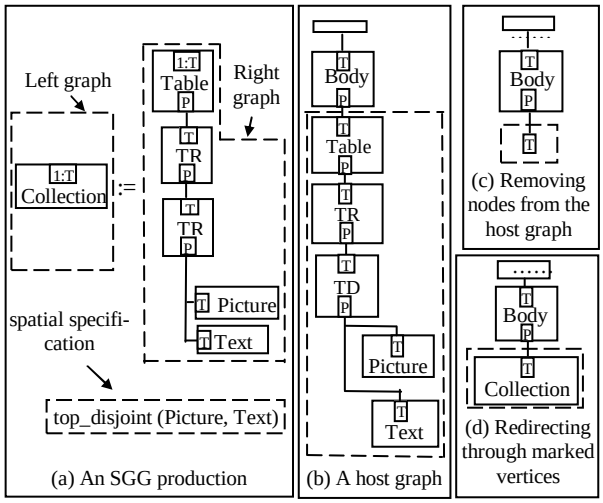


Figure 2: The Spatial Graph Grammar Formalism

Applying a production to a graph (i.e. a *host graph*) is referred to as a graph transformation, which can be classified as an *L-application* (in a forward direction) or *R-application* (in a reverse direction). In particular, an R-application replaces the right graph in a production with the left graph in a host graph. In the page segmentation, we are using R-applications (i.e. the parsing process) to analyze the information organization.

The spatial graph grammar (SGG) [10] introduces spatial notions to the abstract syntax. In the SGG, nodes and edges, together with spatial relations, construct the precondition of a production application. The distinct spatial capability in the SGG allows the designer to formalize design patterns from both the DOM structure and the visual layout. For example, in the above example, the fact that the picture should be placed on top of the textual description can be defined by a spatial specification in an SGG production in Figure 2(a). Our current implementation of grammar induction does not support spatial induction, and designers need to manually revise the induced productions to add spatial configurations.

Due to the multi-dimensional nature of graphs, some mechanism is needed to establish relationships between the surrounding of a redex (i.e. a sub-graph in the host graph which is isomorphic to the right graph in an R-application) and its replacing graph in the host graph [14]. Inherited from the Reserved Graph Grammar [19], the SGG addresses the embedding issue by the *marking* technique. In the SGG, a node has a two-level structure: the node itself and the small rectangles embedded in the node called *vertices*. In order to identify any graph elements that should be reserved during the transformation process, we mark each isomorphic vertex in a production graph by prefixing its label with an integer unique in the node. For

example, in Figure 2(a), vertex *T* in node *Table* is marked. Figure 2(b) gives a host graph, in which a redex is highlighted within a dotted rectangle. Since vertex *T* in node *Table* in the redex has its isomorphic vertex marked in the right graph, it is preserved during the graph transformation as shown in Figure 2(c). Then, the preserved vertex facilitates to embed the newly created node *Collection* to the host graph in Figure 2(d).

3.2. Grammar Induction

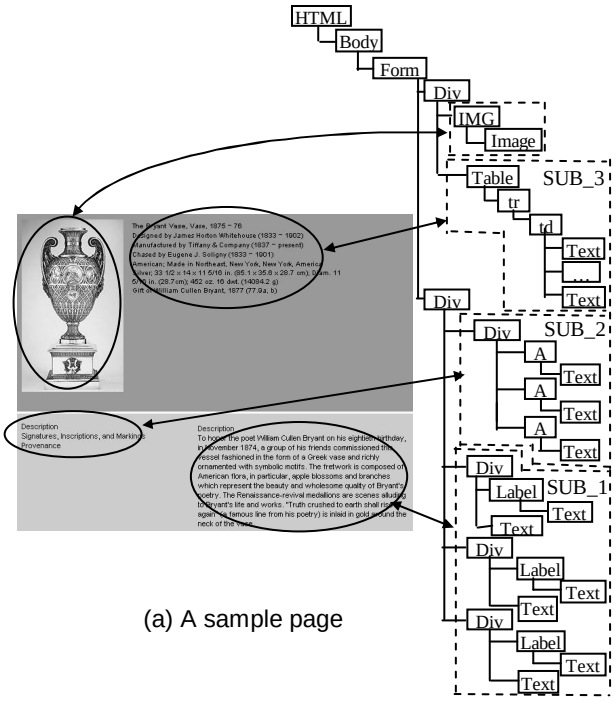
The graph grammar induction system used is a derivative of the SUBDUE Grammar Learner [2]. The SUBDUE induction process is based primarily on the idea of graph compression. Graphs can be compressed in a similar manner to file compression. Like popular file compression techniques, the compression process looks for common substructures within the data, i.e., the graph, and compresses the data by recording the substructure and replacing all instances by a marker. As substructures are captured, this process models a simple context-free grammar induction process. Like string-based data compression, graph-based data compression relies on a substructure matching technique to find instances within the data. However, unlike string-based data compression, for exact sub-graph matching, graph-based data compression requires an exponential runtime graph matching algorithm. Since sub-graph size is generally constrained to less than half of the originating graph and Web page graphs are relatively small, the runtime for the graph matching process has limited impact on the grammar induction process.

4. Grammar Construction

This section discusses the grammar construction based on grammar induction by going through a case study.

4.1. A Case Study

Web designers often have a design pattern to organize and present information when creating a Web page. Such a pattern helps keeping a consistent layout across the whole Web site. Consider an example that the metropolitan museum of arts (http://www.metmuseum.org/) has thousands of artworks, exhibited online using a common design pattern. As illustrated in Figure 3(a), an artwork is presented in a top-bottom style. The top part presents a picture of the artwork at the left side and a brief description at the right side. At the bottom, the left side provides a list of links and the right side presents the detailed information. The corresponding DOM tree is presented in Figure 3(b), in which the top part is separated from the bottom one by using the *Div* elements. The following subsection describes the grammar induction process based on the above example.



(a) A sample page

(b) The DOM structure

Figure 3: A Case study

4.2. Deriving a Graph Grammar

In order to capture domain specific aspects, it is desirable to construct a graph grammar targeting a specific Web site. The grammar construction starts from an induction process, which transforms the sample Web pages into their respective graph equivalents, i.e. essentially the DOM structures for the Web pages, as shown in Figure 3(b). In a Web page's graphical representation, nodes represent HTML tags (such as *TABLE* and *P*) and information blocks (such as *Text* and *Image*); undirected edges denote DOM structural relations.

These DOM graphs are input as a set to the induction processor, which produces a graph grammar. As the DOM structures are trees, the induction process generally produces optimal results. The resulting graph grammar is comparable to a DTD specification for XML documents based on the given DOM structures. This graph grammar can then be reviewed by a human designer for correctness and revision, if necessary, to enhance the grammar's specification. The graph grammar is then used to specify the way of structuring Web pages and derive the semantic structure of a given Web page by parsing its DOM structure.

As an example, the tree shown in Figure 3(b) is given to the induction processor to produce the graph grammar in Figure 4. As the induction process is destructive on the given graph set, the production rules induced for the grammar are applied to compress the graph at the end of each induction. The first induced rule, Figure4 (a), was found to have the highest cost substructure. The second

induced rule, Figure 4(b), is a recursive rule for the first rule. This second rule demonstrated how several instances of a substructure could overlap on a single common node. The induction processor found that instances of the substructure overlapped on a single node—a "*Div*" node—and created the recursive rule to handle the overlap. In practice, the parser would apply the first rule to generate the "*SUB_1*" node and then apply the second rule as needed to process other overlapping instances. The next four production rules provide similar induction solutions. Figures 4 (c) and (d) respectively demonstrate the next highest common substructure and its recursion. Figures 4 (e) and (f) respectively demonstrate the last common substructure and its recursion. These rules also show the independence between the discovered substructures - the latter rules do not include nodes generated in prior rules. The last induced production rule, Figure 4(g), is the "*root*" production rule. This rule ends the induction process by using the remaining graph as a final substructure. For the given graph, this "root" rule finally shows the dependence on the previous rules.

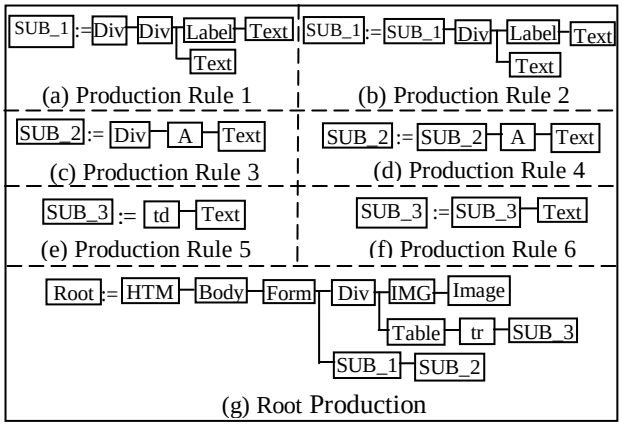


Figure 4: An induced graph grammar

4.3. Revision and Refinement

The induction process looks at a graph in terms of how to best compress it based on a cost calculation, such as minimum description length or size. However, the induction processor lacks the knowledge of applications. Therefore, designers need to elaborate the induced productions.

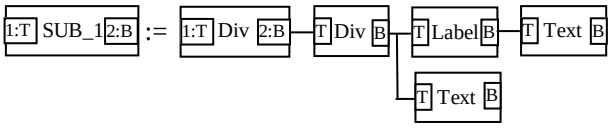


Figure 5. A refinement of Production 1 in Figure 4

Since the induction process cannot automatically identify marked vertices, human designers have to define marked vertices. For example, the induced Production 1 in Figure 4 needs to be refined to the production in Figure 5 by manually defining marked vertices. In addition to marked vertices, designers can also manually add spatial configurations to the induced productions.

The designer can also elaborate the induced productions by introducing domain semantics to the productions. For example, the root production in Figure 4 does not divide the top part from the bottom part. So the designer can divide the root production to three individual productions as in Figure 6. Productions 7 and 8 define the top part and the bottom part respectively, and Production 9 combines the top and bottom parts together.

5. Page Segmentation

5.1. Deriving the content organization

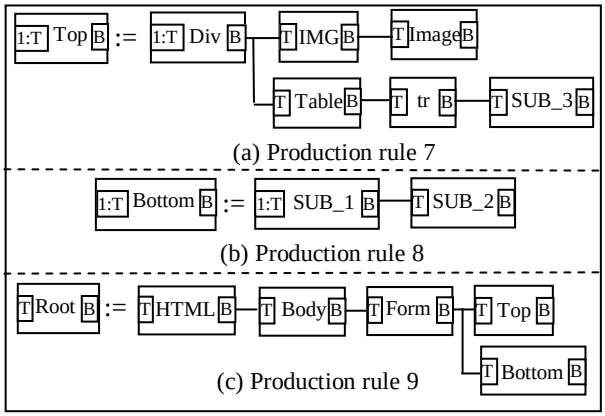


Figure 6. Fine-grained productions

Productions 1 to 6 in Figure 4 and Productions 7 to 9 in Figure 6 construct a graph grammar, which specifies the derivation of semantic structures for the template in Figure 3. In a graph grammar, we classify nodes into HTML nodes and information nodes. An HTML element is considered to be an HTML node and any other node is considered to be an information node. An information node can be further classified as terminal information nodes and non-terminal ones. Each production defines a local composition of information. More specifically, the left graph in a production includes one and only one non-terminal information node (others are contextual nodes), and the right graph contains several (non-)terminal information nodes and HTML nodes. Then, the non-terminal node in the left graph is made up of the (non-)terminal nodes in the right graph. For example, Figure 7(a) gives a host graph, where a redex matching the right graph of Production 1 is enclosed in a dotted rectangle. The application of Production 1 indicates that object SUB_1 consists of two single information objects, i.e. Text. Figure 7(b) is obtained after applying Production 1 to the host graph. Based on Production 1, we derive an information organization as shown in Figure 7(c).

A recursive production, in which the left graph and the right graph include the same type of non-terminal information nodes, is used to recognize the information block,

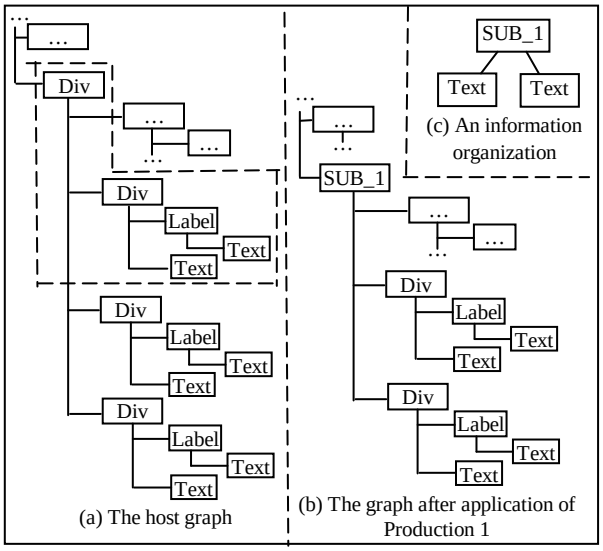


Figure 7. A semantic derivation

which is repeatedly presented with the same pattern. For example, Production 4 in Figure 4 is a recursive production, used to derive the structure SUB_2, which consists of a series of sentences. In the structure SUB_2, Production 3 is first applied to recognize the first sentence and then Production 4 is repeatedly applied to identify the remaining sentences. Correspondingly, Productions 3 and 4 can derive a hierarchical structure of information organization as presented in Figure 8(a). Since the SUB_2 node at the right graph in Production 4 functions as a context node (though it is not a real context node), which connects the previous production application with the next one, we can simplify the structure in Figure 8(a) to Figure 8(b) by excluding context-like nodes. Then, Figure 8(b) can be further simplified to Figure 8(c) by removing intermediate nodes.

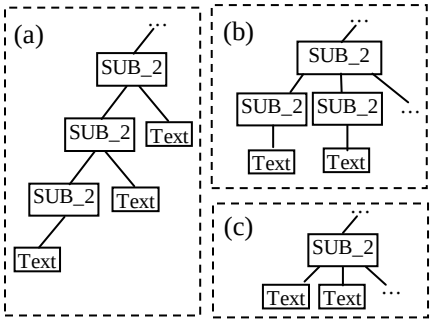


Figure 8. Recursive applications

Corresponding to the example page in Figure 3, a complete hierarchical structure of information organization detected through a parsing process is presented in Figure 9, in which the leaf nodes are atomic information blocks and intermediate ones are composite information blocks.

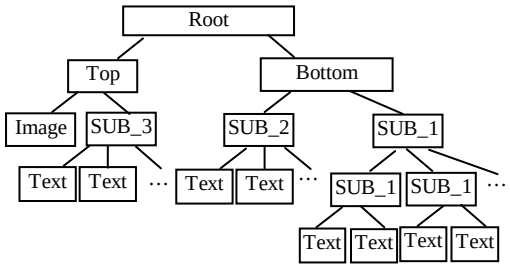


Figure 9. The information organization

5.2. An optimized parsing algorithm

In our grammar-based approach, the time complexity of the grammar parser is critical to the overall performance. Based on the hierarchical nature of information organization, we can enforce on productions an application order, which can eliminate the backtracking in the parsing process. More specifically, a composite information block at a high level is made up of composite/atomic information blocks at a low level. Correspondingly, the parser needs to proceed in a bottom-top fashion, which recognizes low-level information blocks before high-level information blocks. In other words, if information block B_2 transitively or directly includes information block B_1, Production X defined to recognize B_1 must be applied before Production Y defined to recognize B_2. This observation motivates an optimized parsing algorithm, which automatically sequences productions and then applies productions to Web pages in a certain order. The optimized parsing algorithm is running in polynomial time.

Given a production p≡(L,R), in which L represents the left graph and R indicates the right graph, we define the following node sets:

- *LG(p)*: the set of nodes in the left graph;
- *RG(p)*: the set of nodes in the right graph;
- *L/RG(p)*: the set of non-context nodes in the left graph; and
- *R/LG(p)*: the set of non-context nodes in the right graph.

Given a node set N, *Type(N)* denotes the set of node labels. For example, given Production 1 in Figure 4, $Type(LG(P_1))=\{SUB\_1\}$ and $Type(RG(P_1))=\{Div, Label, Text\}$.

If the application of Production P_j depends on the object O_i, which is created by the application of Production P_i, P_i must be applied before P_j. In other words, P_j depends on P_i. Such a dependent relation can be formalized in Definition 1.

Definition 1: Production P_j is *dependent* on Production P_i if $Type(L/RG(P_i))\cap Type(RG(P_j))\neq\emptyset$.

Based on Definition 1, a recursive production depends on itself, i.e., self-dependent. For example, given Production 2 in Figure 4, $Type(L/RG(P_2))=\{SUB\_1\}$ and $Type(RG(P_2))=\{SUB\_1, Div, Label, Text\}$. Therefore,

Type(L/RG(P_2))∩Type(RG(P_2))≠Ø, which indicates that P_2 is self-dependent. The application of a self-dependent production may trigger another round of applications of itself. Semantically, a self-dependent production discloses a nested composition.

Based on the dependent relations among productions, we generate a dependency graph through the following steps:

- Each production is represented as a node; and
- if Production P_j is dependent on Production P_i, a directed edge is inserted to connect from P_i to P_j.

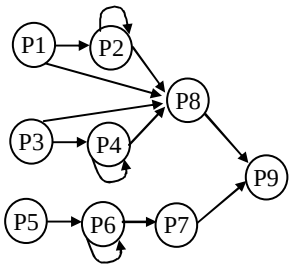


Figure 10. A dependency graph

A dependency graph gives a global view on dependent relations among productions. In a dependency graph, a node with an incoming edge indicates that the application of its corresponding production depends on the application of other productions. On the other hand, a node with an outgoing edge implies that the application of its corresponding production may trigger the application of another production. Figure 10 illustrates a dependency graph for the graph grammar in Figures 4 and 6.

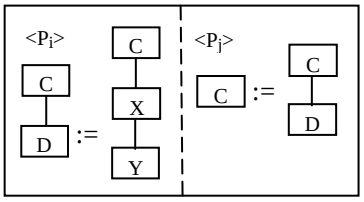


Figure 11. A cycle

Based on the dependency graph, we can assign each production an application priority. In a dependency graph, a production without any incoming edge has the highest application priority and the application priority of productions is decreased along the directed edges. In order to assign each production a unique application priority, the dependency graph must be cycle-free. However, context elements and self-dependent productions can introduce cycles to a dependency graph. For example, two productions in Figure 11 depend on each other, which can form a cycle. The following steps are used to remove cycles and generate a cycle-free dependency graph:

1. Detect all the cycles in the dependency graph;
2. All of the productions involved in a cycle form a cycle set;
3. If two cycle sets have common productions, merge those two sets;
4. Repeat Step 3 until all sets are disjoint;
5. Each disjoint set indicates a super-node;
6. Redirect the incoming and outgoing edges of a production in a cycle set to its corresponding super-node, and then delete the production from the set;
7. Repeat Step 6 until all productions in cycle sets are deleted.

```
AssignLevels(Graph G)
{
  Level ← 1;
  While (G is not empty) {
    for any (super-)node without an incoming edge
      the level of its corresponding production(s) ← Level;
    Remove all nodes without incoming edge and their outgoing edges;
    Level ← Level+1;
  }
}
```
Figure 12. Calculating the application priority of each production

In a cycle-free dependency graph, a single node denotes a production and a super-node presents a set of productions. Based on the cycle-free dependency graph, Figure 12 presents an algorithm to calculate the application priority of each production. A production at level 1 has the highest application priority. All productions in a super-node have the same application priority.

```
Parsing (HostGraph G)
{
    currentLevel ← 1;
    while (currentLevel ≤ MaxLevel) {
      for (any Production p in the currentLevel) {
          for (any node n ∈ G) {
              if p is a single production
                  TreeComparision(G, n, p); }

              if p is a super-production{
                  for (any production q in the p)
                      TreeComparision(G, n, p); }
          }
      }
      currentLevel ++;
    }
}
```
Figure 13. A parsing algorithm

Figure 13 gives a parsing algorithm for a set of ordered productions. The parsing process initially applies production(s) in the level 1 (i.e. the highest application priority), and then proceeds with productions in the next level after no production in the current level is applicable to the host graph. If multiple productions are at the same level, we arbitrarily select a production, since productions in the same level are neither conflicted under the confluent condition nor dependent on each other. Informally, the confluence condition indicates that different sequences of production applications can achieve the same result. Many real world applications can be specified through confluent grammars, evidenced by over 1000 pages of specifications using PROGRES [8].

Theorem 1: Under the confluent condition, the parsing time is bounded at $O(MN^2)$, where M represents the number of productions, and N denotes the number of nodes in the host graph being parsed.

Proof: With a production *p*, we need to traverse the host graph and apply each node to *p*. If node *a* in the host graph matches node *b* in the right graph of *p*, finding in the host graph a sub-tree, which matches the right graph, needs $O(N)$ in the worst case. Therefore, it needs $O(N^2)$ to apply each node to a production. With a total of *M* productions, the time is $O(MN^2)$.

6. An Adaptive Layout

Most heuristic approaches take a straightforward partial screen dump from the source page as an adaptive layout for mobile devices. Instead, our approach can perform a fine-grained page segmentation, which allows designers to flexibly re-organize and re-present the information in an adaptive layout. Related information, which may not even be displayed in proximity in the original Web page, can be grouped together on a small screen. Based on the fine-grained segmentation, this paper presents a multi-page layout while we can easily extend our approach to other styles of adaptive layout, such as a zoom-based presentation [3] or a narrow-column layout [13].

The multi-page presentation separates each detected topic into a small page, called a *sub page*. In general, each sub page presents semantically related information, suitable for browsing on a small screen with the minimal scrolling. A complicated web page in general includes a large number of different topics. Therefore, it is necessary to provide an overview, e.g. a table of contents [1], in a multi-page presentation. A table of contents summarizes different topics in the original Web page and provides quick access to a specific topic. For example, corresponding to the hierarchical structure in Figure 9, Figure 14(a) presents a table of contents. After a user clicks on a specific link in the table of contents, the user is directed to a specific subpage for a detailed reading as in Figure 14(b).

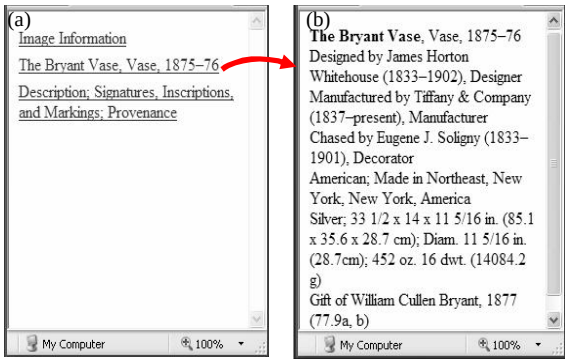


Figure 14. A sub-page layout

The splitting procedure traverses the derived hierarchical structure from top to bottom, and determines which nodes are placed in the same sub page based on the size of each node:

1. If the node is an intermediate node and it is larger than the screen size, the information enclosed in this node is too large to be displayed in a single sub page. This node should be further divided.
2. If the node is an intermediate node and it has a similar size as the screen, the information enclosed in the node is displayed in a single subpage.
3. If an intermediate node is smaller than the screen size, traverse its siblings and combine this node with its siblings to construct a single subpage until the last sibling is reached or the combined size is larger than the screen.
4. If a leaf node is reached, the information enclosed in the node is displayed in a single subpage.

7. Related work

In our approach, graph grammar induction is an important step for the adaptation of multimedia documents. The induction process allows for a wide variety of web domains to be easily adapted and maintained, and significantly reduces the setup costs over a manual system. Additionally, the induction process adds value to the process by providing structural insight into the domain. Important structures that may be overlooked in a manual process are captured from the sample set. Our approach is the only known approach that applies grammar induction to generate adaptive layouts

CSS [16] associated with HTML provides flexible markups for multiple alternative layouts. Constraint-based approaches [4, 11] are capable of dynamically generating multimedia presentations, which are adaptive to the change of media contents, display environment and user intention. Zhang *et al.* [21] proposed a grammatical approach for adaptive layouts by specifying a high level structure and spatial relations among information objects through a graph grammar. Those approaches provide powerful authoring tools to specify adaptive layouts when the viewing condition is dynamically changed. However, redesigning existing pages would be time-consuming. Instead of providing an authoring tool to support adaptive presentations, our approach adapts the existing Web pages for mobile devices.

Many heuristic approaches [5, 6, 9, 12] use HTML structural tags (like Table) to partition a Web page. Kaasinen *et al.* [9] proposed an HTML/WML conversion proxy server, which converts HTML-based Web contents to WML by mapping HTML structures to WML specifications. Buyukkokten *et al.* [5, 6] divided a Web page into several semantic textual units with the help of HTML tags, e.g. the Tag *P* may serve as the boundary between two semantic textual units. This method, however, only focused on texts without supporting graphics. The SmartView [12] uses a thumbnail to provide a visual overview of a page and partitions a page into logic units according to table tags. Opera [13] provides a small-screen rendering technology, which stacks Web contents vertically to

avoid horizontal scrolling. This method may falsely separate closely related contents or combine unrelated information together. Yang *et al.* [17] evaluated the visual similarities of HTML contents, detected the pattern of visual similarity, and generated a hierarchical representation of the HTML page. Chen *et al.* [7] first divided a Web page into several high level information blocks according to their sizes and locations, and then identified explicit and implicit separators inside each high level block.

Visual language formalisms have been used to analyze the information organization underlying a Web page, such as Web queries [20] and adaptive layouts [10]. However, they do not use grammar induction techniques technique and thus require human expertise to manually design grammar rules.

8. Conclusion

Unlike heuristic approaches, our approach is built on a formal basis of the Spatial Graph Grammar formalism. The derivation of an information organization underlying a Web page is essentially a parsing process, which incrementally exploits the hierarchical information composition from bottom to top. Our approach uses the grammar induction technique to automate the grammar construction. The automation process significantly reduces the manual effort of designing a graph grammar, which increases the applicability of our approach. Based on the hierarchical nature of information organization, we have designed an optimized parsing algorithm, which can reach polynomial time under the confluent condition. Based on the derived hierarchical structure, we can re-organize and present related information on a small screen. In future work, we will consider providing alternative presentations for dynamic contents in Web pages.

9. References

[1] H. Ahmadi and J. Kong, "Efficient Web Browsing on Small Screens", *Proc. ACM AVI'2008*, 2008.

[2] K. Ates, J. Kukluk, L. Holder, D. Cook, and K. Zhang, "Graph Grammar Inference on Structural Data for Visual Programming", *Proc. IEEE ICTAI'06*, 2006.

[3] P. Baudisch, X. Xie, C. Wang, W. Ma, "Collapse-to-Zoom: Viewing Web Pages on Small Screen Devices by Interactively Removing Irrelevant Content", *Proc. UIST*, 91-94, 2004.

[4] A. Borning, R. K. Lin, and K. Marriott, "Constraint-based Document Layout for the Web", *Multimedia systems*, 8, pp.177-189, 2000.

[5] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, "Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices", *Proc. WWW'01*, 2001.

[6] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke, "Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones", *Proc. ACM SIGCHI'*, pp.213-220, 2001.

[7] Y. Chen, W. Y. Ma, and H. J. Zhang, "Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices", *Proc. WWW'03*, pp. 225-233, 2003.

[8] T. Fischer, J. Niere, L. Torunski, and A. Zündorf, "Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java", *Proc. Theory and Application to Graph Transformations*, LNCS 1764, pp. 296-309, 1998.

[9] E. Kaasinen, M. Aaltonene, J. Kolari, S. Melakoski, and T. Laakko, "Two Approaches to Bringing Internet Services to WAP Devices", *The International Journal of Computer and Telecommunications Networking*, 33, pp. 231 – 246, 2000.

[10] J. Kong, K. Zhang, and X.Q. Zeng, "Spatial Graph Grammar for Graphic User Interfaces", *ACM Transactions on Human-Computer Interaction*, Vol.13(2), pp. 268-307, 2006.

[11] K. Marriott, B. Meyer, and L. Tardif, "Fast and Efficient Client-side Adaptability for SVG", *Proc. WWW'02*, pp. 496-507, 2002.

[12] N. Milic-Frayling, and R. Sommerer, "SmartView: Flexible Viewing of Web Page Contents", *Proc. WWW*', 2002.

[13] Opera Software ASA, http://www.opera.com/products/mobile/smallscreen, 2008.

[14] J. Rekers and A. Schürr, "Defining and Parsing Visual Languages with Layered Graph Grammars", *Journal of Visual Languages and Computing*, 8(1), pp.27-55, 1997.

[15] G. Rozenberg and E. Welzl, "Boundary NLC Graph Grammars - Basic Definitions, Normal Forms, and Complexity", *Information and Control* 69, pp.136-167, 1986.

[16] W3C, Cascading Style Sheets (CSS), 2005.

[17] Y. D. Yang and H. J. Zhang, "HTML Page Analysis Based on Visual Cues", *Proc. 6th International Conference on Document Analysis and Recognition*, pp. 859-864, 2001.

[18] S. P. Yu, D. Cai, J. R. Wen, and W. Y. Ma, "Improving Pseudo-Relevance Feedback in Web Information Retrieval Using Web Page Segmentation", *Proc. WWW'03*, pp.11-18, 2003.

[19] D. Q. Zhang, K. Zhang, and J. Cao, "A Context-Sensitive Graph Grammar Formalism for the Specification of Visual Languages", The *Computer Journal*, (44)3, pp.187-200, 2001.

[20] Z. Zhang, B. He, and K. C.-C. Chang, "Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax", *Proc. 2004 ACM SIGMOD*, pp.107-118, 2004.

[21] K. Zhang, J. Kong, M.K. Qiu, and G.L. Song, "Multimedia Layout Adaptation Through Grammatical Specifications", *Multimedia Systems*, 10(3), pp.245-260, 2005.