A Survey of Automated Layout Techniques for Information Presentations

Simon Lok and Steven Feiner
Dept. of Computer Science
Columbia University
1214 Amsterdam Ave.
New York, NY 10027 USA
{lok,feiner}@cs.columbia.edu

ABSTRACT

Layout refers to the process of determining the sizes and positions of the visual objects that are part of an information presentation. *Automated layout* refers to the use of a computer program to automate either all or part of the layout process. This field of research lies at the crossroads between artificial intelligence and human computer interaction. Automated layout of presentations is becoming increasingly important as the amount of data that we need to present rapidly overtakes our ability to present it manually. We survey and analyze the techniques used by research systems that have automated layout components and suggest possible areas of future work.

Categories and Subject Descriptors

H.5.2 [**HCI**]: User Interfaces—*Screen Design*; I.2.1 [**AI**]: Applications and Expert Systems; I.3.6 [**Computer Graphics**]: Methodology and Techniques

Keywords

Automated Layout, Intelligent User Interfaces, Knowledge-Based Graphics, Information Visualization, Artificial Intelligence, Human Computer Interaction

1. INTRODUCTION

Effective layout is one of the most important aspects of creating an information presentation. By *layout*, we mean both the process of determining the position and size of each visual object that is displayed in a presentation, and the result of that process. We use the term *presentation* to refer to material that is intended to be viewed and manipulated by people; for example, graphical or textual user interfaces (UIs), World Wide Web documents, and even conventional newspapers and magazines. Layout is one component of a presentation's design, and must complement other decisions that determine the number and nature of the visual objects that are laid out: the information to be communicated and its format. By *format*, we mean how the visual objects are realized (e.g., as text, graphics, or UI widgets), and their attributes (e.g., color, texture, or font).

A presentation's layout can have a significant impact on how well it communicates information to and obtains information from those who interact with it. For example, the importance of individual objects can be emphasized or deemphasized, and the relationships between objects can be made more or less clear. In the case of UIs, layout determines in part the time, effort, and accuracy with which tasks can be accomplished. A well laid out presentation can visually guide the viewer to infer correct relationships about its objects (e.g., to view a linear presentation from beginning to end) and to accomplish tasks quickly and correctly (e.g., by minimizing the distance between objects to be manipulated sequentially), increasing the presentation's effectiveness.

The vast majority of layouts created today are done "by hand": a human graphic designer or "layout expert" makes most, if not all, of the decisions about the position and size of the objects to be presented. Designers typically spend years learning how to create effective layouts, and may take hours or days to create even a single screen of a presentation. Designing presentations by hand is too expensive and too slow to address situations in which time-critical information must be communicated. Recognizing this problem has given rise to work in automated generation of graphical presentations and UIs [12]. This research has addressed the creation of objects to be presented and the determination of their attributes (what we have referred to before as format), and the layout of these objects.

In this paper we provide a survey of work on automated layout for presentations, recognizing that the objects to be laid out may be derived wholly or in part from automated generation either before or after layout. To focus on the general problem, we restrict our study to not address the very active field of graph layout [1]. Not only has this field been extensively studied, with its own annual conference [13], but many of the issues with which it is concerned are specific to problems caused by the explicit visual representation of graph edges (e.g., minimizing edge crossing [10, 54]). We also note that "automated layout" is a term that is commonly used to refer to automated circuit layout for VLSI chip fabrication [17, 29] and to automated placement of pieces to be cut from a bolt of cloth used to produce clothing [39]. Unlike presentation layouts (including graph layouts), these layouts are designed to meet the requirements of a fabrication process (e.g., to minimize chip area and interconnect length in VLSI [3]), rather than to make them un-

This work is supported in part by the National Science Foundation Digital Libraries Initiative Phase 2 as part of the PERSIVAL project at Columbia University under award IIS-98-17434.

SmartGraphics '01 Hawthorne, NY USA

derstandable to humans. While we cover neither of these areas, we note that they employ some techniques that may contribute to automated layout of presentations (e.g., general constraint solvers) and other that may not (e.g., bin-packing techniques [15] that result in minimal area layouts at the expense of maintaining visually obvious relationships among objects).

In the remainder of this paper, we first discuss some of the simple approaches to presentation layout used in current commercial software in Section 2. Next, we focus on the two methods that have been adopted by research systems that address automated presentation layout: constraints and learning. As described in Section 3, most research layout systems assume that a layout can be described by a set of constraints. Much of this work addresses applying constraint solvers and developing new ones. In addition, many researchers have focused on how to determine or extract the right constraints as well as how to to express them to a computer program. Systems that are not constraint-based generally fall into the category of learning systems, which we discuss in Section 4. These systems are sometimes trained by experts and other times by the user as the system assists in the layout process. In Section 5, we introduce some of the issues involved in evaluating presentation layouts. Finally, Section 6 presents our conclusions and ideas for future work.

2. SIMPLE TECHNIQUES

Almost all contemporary user interfaces are built with a UI toolkit (e.g., Sun JFC/Swing [23], Microsoft Foundation Classes [38], and their ancestors, such as Xtk [37] or Tk [44]) that provides basic functionality, such as creating buttons and windows. Toolkits include *layout managers* (also known as *geometry managers*) to assist the UI designer (programmer) in designing the layout of objects in a managed container, without having to specify the absolute position and size of each object. Layout managers allow the programmer to say, in effect, "add button" or "add button to this part of the window," and optionally specify additional numeric constraints. This makes it possible to create layouts that adapt to changes in the container's size.

A layout manager chooses positions and sizes at run time for the objects that it controls, governed by a set of constraints imposed by a simple layout policy built into the manager and parameters specified by the programmer. Typical layout policies include strict horizontal (row) or vertical (column) layout; row-major or column-major layout in which objects wrap to the next row or column to avoid exceeding the managed container's bounds; border layout in which objects can be specified to reside in the north, south, east, west, or center of the managed container; and grid layout in which objects are specified to reside at one position (or straddle multiple positions) in a programmer-specified 2D grid. Programmer-specified parameters indicate preferred, minimum, and maximum widths and heights of objects; and spacing, both between objects and between objects and the container's edges. Managed containers can be nested inside of other managed containers, and treated just like any other objects.

A programmer designs a parameterized layout as a hierarchy of managed containers, chosen for their layout policies, and further constrained by programmer-specified parameters. Thus, a layout manager does not actually design a layout, but rather instantiates a layout at run time from the structure and parameters specified by the programmer. Designing a simple layout (e.g., four buttons displayed in row-major order) is easy for a programmer, but designing a complex hierarchical layout, while possible, is tedious and difficult, especially if it needs to behave robustly when resized. The popularity of the layout-manager approach stems primarily from the ease of implementing the managers themselves and the relative ease with which they can be used by programmers to define simple parameterized layouts. In addition, because the final layout is determined at run time, the presentation can work well under a wide range of possible window sizes, with lower bounds on usability imposed by the minimum effective sizes of the objects being laid out.

Commercial word-processing and presentation systems intended primarily for sequential presentations (e.g., Microsoft Word, Publisher, and PowerPoint, Quark Express, and LaTeX), provide a set of preauthored style "templates" (and the ability to create new ones) that can be applied to existing material; for example, by matching "markup" tags present in the material to be processed with corresponding tags in the template. In comparison with the parameterized layouts of UI toolkit layout managers, most of these template-based systems are simpler. They emphasize the format of the material to be presented, relying on the order in which objects are specified as input to directly determine the order in which formatted objects appear in the presentation, with the exception of their handling of floating objects. *Floating* objects, such as tables or figures, can move relative to surrounding objects (typically text). Typically, a set of rules is used to determine the final position of the floating object; for example, "If the height of the object is more than 60% of the height of the page, then put the object on a separate page; otherwise, put the object at the nearest paragraph break." However, users of these systems often move floating objects around by hand to guide or overrule simplistic placement policies.

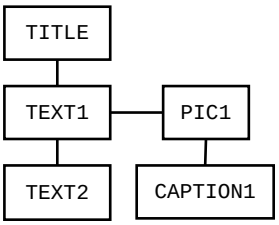
In the following sections, we discuss the two techniques that have been explored in automated layout systems: constraint satisfaction and learning.

3. CONSTRAINT SATISFACTION

The vast majority of research in automated layout to date focuses on constraint-based methods (e.g., [59, 4, 14, 20, 26, 19, 61, 42]). The idea that layouts can be represented as a set of constraints is very intuitive. For instance, consider the constraint relationships depicted in Figure 1. The goal of a constraint-based automated layout system is to take such a constraint network and generate a set of positions and sizes for each of the components in the network. Any constraint-based automated layout system can be characterized by the kinds of constraints it uses; how they are described, obtained, and resolved; and how the system addresses constraint inconsistencies, loops and other hazards that might prevent the solution from converging. In the rest of this section, we explore these issues.

3.1 Types of Constraints

In a constraint-based automated layout system, most constraints can be classified as either abstract or spatial. By *abstract*, we mean that the constraint describes a high-level relationship between two components that are to be included in the layout (e.g., "TEXT1 REFERENCES PIC1"). In contrast, *spatial* constraints enforce position or size restrictions on the components (e.g., "CAPTION1 BELOW PIC1"). Spatial constraints can be fed directly into a constraint solver for resolution, whereas abstract constraints must be processed and reduced to spatial constraints before the layout can be realized. The process of generating spatial constraints from the abstract constraints is often the most challenging part of creating an automated layout system.



(a)

TITLE ABOVE TEXT1
TITLE FULL PAGE WIDTH
TEXT1 LEFT OF PIC1
CAPTION1 BELOW PIC1
TEXT2 BELOW TEXT1

(b)

TITLE DESCRIBES TEXT1
TITLE IS IMPORTANT
TEXT1 REFERENCES PIC1
CAPTION1 DESCRIBES PIC1
TEXT2 FOLLOWS TEXT1

(c)

Figure 1: (a) A network of constraints that might be used in an automated layout system. (b) A set of spatial constraints that express the constraint network. (c) A set of abstract constraints that express the same relationships.

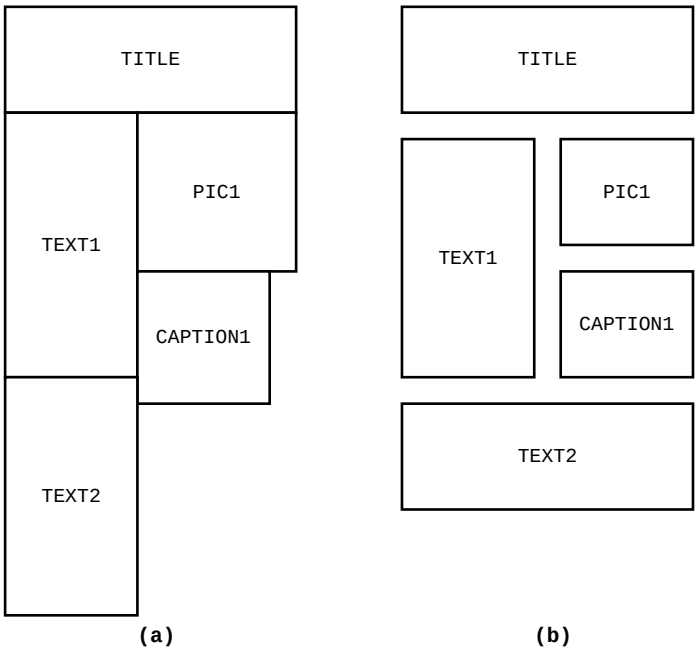
The choice of employing abstract or spatial constraints depends on the nature of the layout that the system is designed to generate. Many research systems employ both abstract and spatial constraints because they are general multimedia presentation tools [11, 61, 14]. Other research initiatives are embodied in the form of libraries such as subArctic [20, 18] and the Garnet toolkit [42] that include extensions of "layout manager" type functionality with simple spatial constraints. Although automated layout systems for more limited environments can create effective layouts using only spatial constraints [27], the same can not be said for attempting to employ abstract constraints without spatial constraints. Figure 2 depicts what the difference in output might be between a system that considers only abstract constraints versus a system that takes into account both abstract and and spatial constraints. This issue is discussed further in Section 3.1.2.

3.1.1 Abstract Constraints

Abstract constraints are descriptions of high-level relationships between the various components that are to be placed into the layout. Abstract constraints such as "TEXT1 REFERENCES PIC1" and "TITLE IS IMPORTANT" are sufficiently high-level that content authors can easily specify them and are particularly effective for use in interactive systems because the author of the content needs no additional technical or artistic skill to specify them.

Although one might think that "TEXT1 REFERENCES PIC1" implies that B and C are relatively close to each other in the generated layout, abstract constraints in and of themselves do not specify the position and size of the components of a layout. This is because the mapping between abstract constraints and spatial constraints is performed by a translation component before spatial constraints are passed to the numeric constraint solver. The abstract–spatial constraint translator can choose to map the abstract constraint "TEXT1 REFERENCES PIC1" into *any* set of spatial constraints. This might mean that the PIC1 is placed right next to text TEXT1, or that PIC1 is placed on a different page and some visual cue is left to guide the end user to it from TEXT1.

3.1.2 Spatial Constraints



(a) (b)

Figure 2: (a) A simple layout that can be generated by a system that only considers abstract relationships between components. (b) A layout of the same components where additional spatial constraints are enforced so that each component completely fills a regular grid and leave margins between the elements.

Spatial constraints are relations that directly express the geometric structure of the presentation. For instance, we may wish to force a certain block of text to appear beneath another block of text that the user is assumed to have read first. Another instance of spatial constraints would be to force all objects to occupy a space that is of a certain size or an integral multiple of that size.

There are a number of reasons why we would want to impose spatial constraints. Perhaps the most prevalent reason is to improve upon the visual quality and aesthetics of the presentation. Many early automated layout systems are created from the perspective of computer science and mathematics alone [2]. Such systems tend to treat the problem as a purely theoretical question of tiling and use optimization techniques to find a solution [31]. These kinds of systems will often not take into account simple legibility rules (e.g., text should be placed into columns that run down the entire page rather than having blocks of random size packed onto the screen) and style guidelines (e.g., all captions go beneath their associated figures and spacing between a figure and surrounding text block should be the same everywhere). Figure 2 exemplifies what might happen in a system that employs abstract constraints without spatial constraints. A system that considers only abstract constraints will not be able to generate layouts with the same aesthetic appeal as systems that consider both because the system has no visual restrictions on where components of the layout are placed.

The method by which the components are represented may place some kind of limitation on what kinds of spatial constraints may be used. One such issue that may arise is sometimes referred to as the "Cousins Problem," an example of which is shown in Figure 3. This

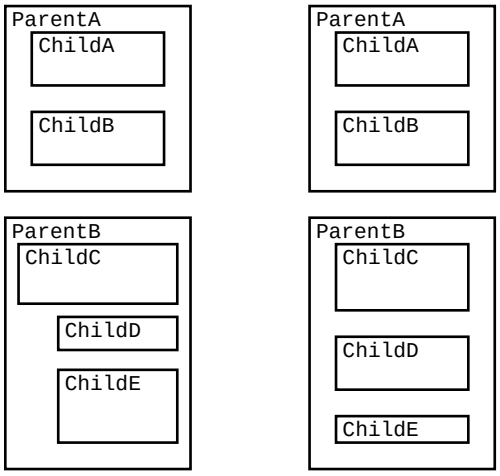


Figure 3: (a) A layout generated by a system that does not enforce spatial relationships between "cousins." (b) A layout of the same components where spatial constraints force all of the children to be aligned.

problem may arise if the data structure in which the components are being stored does not allow referencing one component's children from a different component's child.

It seems intuitive that we would want to use concepts from graphic design to create legible and pleasing output. Some systems enforce the presentation to conform to a grid system, similar to those used to lay out newspapers [40, 21]. In a *grid system*, every screen or page of the presentation is divided into an array of upright rectangles. Each object must occupy one or more complete rectangles. Figure 4 is an example of output from Feiner's GRIDS system [11] which designs layout grids that enforce a consistency between screens or pages of a presentation. One complication with employing grid systems is that a graphical component may need to be cropped, padded with a border or have its aspect ratio changed This is because uniform scaling of the object may not be sufficient to make the object occupy an integral number of grid rectangles.

Automated layout systems for well-defined environments, such as network diagrams or label placement, often employ spatial constraints exclusively [27, 7]. These systems consider issues such as the proximity of the components being placed, the distance between a label and its target, and the possibility of confusing the end user by placing multiple labels that are sufficiently near the same target that the end user doesn't know which label is associated with it. Abstract constraints are often used for formatting labels (larger cities have bigger names), but are generally not used for layout directly.

Some systems allow the user to specify abstract data constraints separately from spatial constraints. This allows for a logical single presentation to have many different "skins," opening the door for a single presentation to be displayed using different media [61]. This is particularly effective for interactive layout systems that might want to maintain a separation between content authors and "layout experts." A similar approach that leverages this concept is to build the spatial constraints into the system, thereby eliminating the need

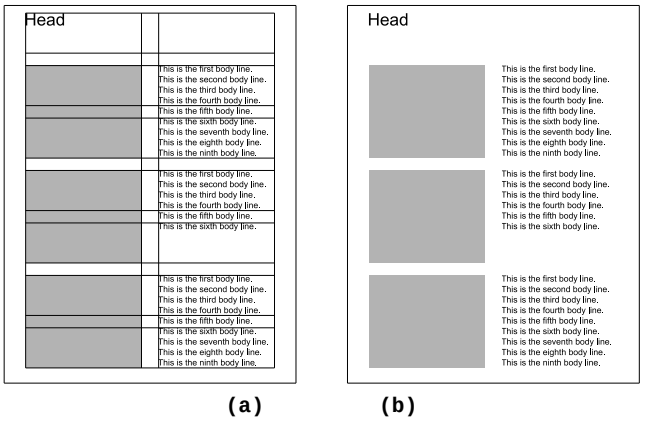


Figure 4: (a) A layout generated by Feiner's GRIDS with the grid showing. (b) The same layout as presented to the user.

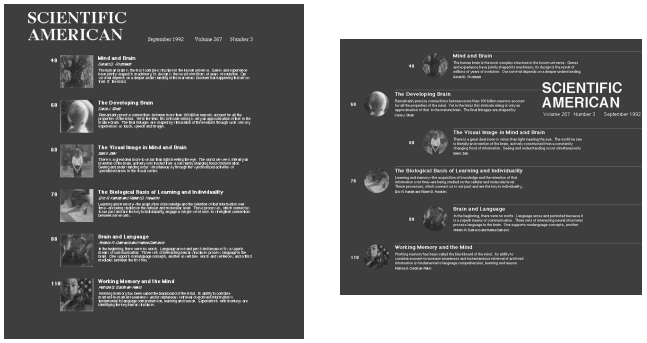


Figure 5: Two layouts generated by Weitzman's system using the same content and abstract constraints with different visual constraints

for human intervention to specify spatial constraints for each layout to be generated. Feiner's GRIDS is an example of a such system [11].

Spatial constraints are sometimes used to increase the efficiency of the constraint solver. For instance, a constraint might be placed on all objects of a certain type that permits them to be resized in only one direction [30]. Similarly, imposing the constraint of a quantized display permits the use of fast fixed point and integer programming techniques when resolving the set of constraints.

3.2 Expressing Constraints

Intuitively it would seem best to define a formal grammar to describe the method by which the constraints are expressed. This approach benefits from being able to leverage a rich body of existing research for manipulating and parsing constraints. A rich grammar might be very flexible and expressive and translate into better layouts [60]. An example of a grammar for use in a layout system is shown in Figure 6. However, powerful and expressive grammars may also be difficult to use. This is especially true of grammars or ontologies that attempt to be extremely general and all-encompassing. In addition, a very complex solver may be necessary to process the information present in such a system. As one might imagine, it is extremely difficult to create a system that de-

```
(Defrule (Make-Article The-Grammar)
  Article -> Text Text Text Number Image
     0         1    2    3     4     5
  (Author-Of 2 1)
  (Description-Of 4 1)
  (Page-Of 4 1)
  (Image-Of 5 1)
  (article-name 0) = r
  (article-image 0) = 5
  :OUT
  (right-of 1 5)
  (top-aligned 1 5)
  (top-aligned 5 4)
  (spaced-below 2 1)
  (spaced-below 3 2)
)
```

Figure 6: A fragment of a rule definition from Weitzman and Wittenburg's system [61].

scribes the set of all possible high-level relationships between components of a presentation, although this has been explored for the use of automatic graphics generation [62].

Another extremely powerful approach is to express the constraints in terms of Boolean predicates [14]. This approach alleviates some of the concerns that arise from the more expressive grammar and relational grammar approaches by limiting the space of what can be expressed. The use of Boolean predicates also eases the process of solving the set of constraints as the input needs little or no translation before being passed to the solver.

3.3 Obtaining Constraints

One of the most important practical issues in implementing a constraint-based automated layout system is determining where to obtain the constraints. Approaches that have been tried range from fully automated to the computer making suggestions.

Many automated layout systems implement abstract constraints by gathering them from structured input data [33, 6, 4, 2]. These systems take tables of numeric data and automatically create presentations. The structure of the data provides all the relationships that are needed to generate the layout. Other systems that are designed for multimedia layout have languages to explicitly specify the abstract constraints to describe relationships such as "author-of," "description-of" and "precedes" between the components [61, 14]. The assumption that input data is readily available in a structured form is becoming increasingly valid because the information that we create is beginning to be stored in more structured formats [57]. Such work is prevalent in the layout modules of automated graphics generation systems [63].

3.3.1 Interactive Specification

Interactive constraint specification systems are also extremely popular, but suffer from the obvious limitation that they require user input. Some systems are designed to help graphically naive content authors create professional quality layouts. Others are meant to reduce the amount of time a graphic designer needs to spend on a layout.

Most systems that take interactive input do so for spatial constraints [56, 19, 4]. This is because it is easy to create graphical user interfaces that allow the user to interactively place or adjust components on the screen. Although providing a graphical user interface to specify abstract constraints is not unheard of, abstract constraints tend not to need adjustment. Roth's SAGE system [47] allows a user to associate database records with visual elements.

Some of the interactive systems require the user to specify the high-level design of the document and then automatically generate the final result [25]. This approach is very useful in situations where the goal is to enable a content author to create layouts without the need for intervention by a "layout expert." Some other systems take the opposite approach where the system produces the initial layout and allows the user to refine it [55]. These systems are more applicable in situations where the goal is to save the amount of time that a graphic designer needs to spend to create the layout.

3.3.2 Automated Extraction

Fully automated constraint extraction is the least explored method of obtaining constraints. Some work has been done in integrating natural language techniques with automated layout [48]. This is particularly effective if natural language generation is being employed to create the content. Since the content generators are computer programs, it is much easier to have the generator send abstract that describes relationships between components as well as markup the text with flags denoting which parts are particularly important.

Another method that has been explored is to extract abstract constraints from the entity relationships found in SQL databases [46]. Unlike the natural language generation system that passes additional information to the layout system, in this approach abstract constraints are derived from data structures that were originally intended for use elsewhere.

3.4 Constraint Solvers

A constraint-based automatic layout system must have some way to resolve the constraints with which it is presented. Formally, automated layout techniques all solve a form of the constraint satisfaction problem (CSP) [35, 34]. Both randomized and deterministic algorithms have been applied to solve the problems in this field. In general, the constraint solvers employed can be categorized as applying either a local or a global methodology.

3.4.1 Local Techniques

Local constraint solvers attack the constraint satisfaction problem bottom-up. This might be compared to inductive reasoning, where a small subset of the universe is first solved. Two routes can be taken to solv the rest of the constraints and create the final presentation. The first approach is to solve many small subsets of the constraints independently and then run a second resolution phase to combine the results. An alternative approach is to iteratively resolve constraints at the border between the constraints that have already been solved and those that have yet to be considered [43].

Using a local-resolution technique can be problematic if the solver encounters local minima [5]. By resolving small subsets first, the solver may make decisions that bring the system to a suboptimal final solution. The advantage, however, is that local-resolution techniques usually execute much faster than global techniques.

3.4.2 Global Techniques

Global constraint solvers attack the constraint satisfaction problem from the top down. Unlike local techniques, global techniques generally do not suffer from the problem of local minima, but require more computation time. To address the issue of additional

computation time, numeric solvers that use iterative approximation techniques have been applied [28]. Randomized computation techniques (e.g., genetic algorithms and simulated annealing) have also been applied for label placement [8].

3.5 Inconsistency Policies

If the set of constraints is sufficiently large, there is a strong likelihood that there will be some problems. In particular, some constraints may contradict others and possibly make the system of constraints unsolvable. A resolution policy must be specified to generate a layout in these cases.

Some systems take a very simple approach to inconsistency by avoiding it. Rather than bogging the system down with inconsistency handling, the grammar used to articulate the constraints is designed in such a way that cycles cannot occur [61].

Another popular method for handling inconsistency is to apply priorities to the constraints [14]. By permitting each constraint to have an inherent priority, the system can make intelligent decisions about which constraints to drop should a contradiction be encountered. A problem can still occur here if two conflicting constraints have the same priority. In this case, the system can use the AI technique of applying a tie-breaking strategy (e.g., first-come first-served or pick one randomly) so that the layout can be generated [43]. Priorities are critical for generating effective layouts in systems where there are complex networks of both abstract and spatial constraints. For example, the enforcement of the grid in a system that employs design grids must take precedence over all other constraints.

4. LEARNING TECHNIQUES

Machine learning has been applied to many automated multimedia authoring systems, including speech synthesis [45] and natural language generation [24], as well as to graph layout [36]. However, most automated layout systems do not leverage the large body of existing work by the AI community in machine learning.

Automated layout systems that do have some form of learning tend to use it during the interactive specification of constraints [41, 4]. These systems do not implement full machine learning systems. Rather, they try to "learn" based on interacting with the user. The Marquise system [41] allows a user to set the system into a training mode where the relative locations of components are demonstrated to the system. Spatial constraints that will be used to generate the layout are then extracted from this interaction with the user. If the constraints cannot be extracted, the system falls back to having the user specify the position explicitly as a LISP function. Borning's [4] ThingLab is similar in that it allows for demonstration of constraints, but also adds the ability to demonstrate animation.

Some recent work in automated graphics generation has also explored the use of learning techniques [64]. Zhou divides the space of rules that need to be acquired for presentation generation into three categories: information learning space, visual learning space and rule learning space. Visual learning space is directly related to spatial constraints, and thus is similar to Myer's and Borning's work. Unlike Marquise and ThingLab however, Zhou's system employs full-strength machine learning that can be fully automated by providing the system with a large dataset of presentations designed by a "layout expert" in addition to the interactive methods seen in other work that employs learning techniques.

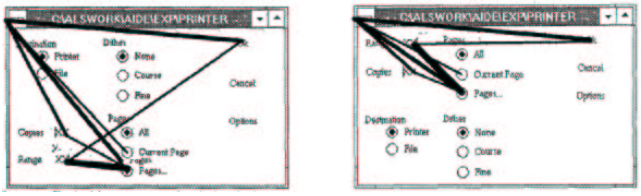


Figure 7: (a) A user interface designed without metric-based evaluation. The lines track mouse movements made by the human user while operating the user interface. (b) A user interface designed with Sears' metric based evaluation methodology. The shorter mouse tracks show that the user does not need to move the mouse as much in order to get the same work done.

5. EVALUATION TECHNIQUES

Whenever a piece of software is used to perform a task that is traditionally believed to be reserved for human "experts," the question that will always be asked is whether or not the computer is as "good" as the human. In the field of layout, "good" may refer to the usability (e.g., whether tasks can be accomplished more quickly, with fewer errors, with greater user satisfaction), as well as the aesthetics of the presentation (e.g., whether end users or graphic designers think that the results look as good as ones produced by humans).

In some sense, all interactive layout generation systems have a module that handles the evaluation of how "good" the layout is: the human user. By using a computer-based evaluation mechanism, we could evaluate the layout automatically without relying (as much) on the user, ideally feeding back the results to redesign a layout that is not deemed good enough.

Evaluation of user interfaces, independent of who or what designs them, has been explored by a number of researchers [58, 51, 53, 9, 22]. The focus of this research has primarily been divided between creating heuristic inferences and quantitative metrics. Heuristics are of course more flexible, whereas metrics are easier to incorporate into computer systems and hence more common in automated evaluation mechanisms.

Sears has explored the application of metrics to automated layout systems [52]. He employed metrics to evaluate how usable an interface is based on the amount of mouse movement that is needed between button clicks. Figure 7 shows screen shots from Sears's system. The additional information provided by these metrics are embodied in the form of spatial constraints. Fitts' Law [32] implies that buttons that are often clicked sequentially should be placed close to each other spatially, since this reduces the time needed to move between them.

Other research has addressed adding evaluation to the automated layout process in the context of user modeling [50]. This work proposes that the interface not only include or exclude certain elements based on the type of user, but that it also change the layout. This information may be gathered at run time—as the frequency of use of different parts of the system change, the layout could be modified to reflect this. An appropriate user model could make it possible to adapt the output for the user. Note, however, the potential for change to create a less, rather than more, effective UI by clashing

with the user's mental map of the user interface.

6. CONCLUSIONS AND FUTURE WORK

We have illustrated the range of research that has been accomplished in the field of automated layout, from simple techniques to research systems. As data presentation needs rapidly increase, the field of automated layout will become increasingly important. In time, we feel it is inevitable that the more powerful techniques found in the research systems will make their way into popular software packages. Reviewing current research, we see a number of rich possibilities for future work.

Integration of natural language techniques with automated layout systems has been explored to some extent, particularly with natural language generation. However, similar work has not been pursued with image and video understanding or speech recognition. It may be possible to extract abstract or spatial constraints for automated layout by applying well known vision or speech recognition techniques to multimedia components of a layout.

Another interesting possibility is considering how to handle constraints that are "wrong." Most systems have a user specifying the constraints in some manner at some point in the layout pipeline. The problem is that the user might simply make a mistake and not really mean what what he or she specified. In a system that has support for ranking constraints by priority, the user might also assign incorrect priorities. Constraints that are automatically extracted opens up even more doors for feeding incorrect information to the system. The use of natural language understanding, image understanding or speech recognition to extract constraints by definition means that there will be some probability of error in the constraints. Enabling an automated layout system that would be capable of handling these kinds of problems might involve applying AI techniques from adversarial game playing [43]. Algorithms from computational biology and genomics may also be applicable because this kind of problem is encountered during gene sequencing [49]. Constraints extracted by an automated process can be verified by running multiple extraction algorithms and having them "vote." A similar approach is employed by an object-recognition technique called the Hough transform [16]. By applying algorithms to defend against constraint error, a system might be made robust enough that errors in the constraints could cause little or no loss of effectiveness in the presentation.

There are other kinds of constraints that might be worthwhile to consider, some of which be obtained through hardware capture of information for user models. For example, real-time systems that automatically generate a user interface would benefit from knowing the user's distance from the display medium (information that was taken into account at the beginning of the layout design process of [11], but not computed automatically). This could be determined from any of a variety of head-tracking technologies. Eye tracking could also be used to extract additional constraints based on where the user is looking. What parts of the display the user has actually seen would be useful information to pass to the automated layout system.

7. REFERENCES

[1] G. D. Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, USA, 1999.

[2] R. J. Beach. Computerized typesetting of technical documents. Technical report, Dept. of Computer Science, University of Waterloo, Waterloo, ON, Canada, 1977.

[3] S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Information Processing Letters*, 25(4):263–267, June 1987.

[4] A. Borning and R. Duisberg. Constraint-based tools for building user interfaces. *ACM Transactions on Graphics*, 5(4):345–374, Oct. 1986.

[5] A. H. Borning. *Thinglab—A Constraint-Oriented Simulation Laboratory*. PhD thesis, Stanford University, July 1979. Also available as Stanford Computer Science Department report STAN-CS-79-746 and as XEROX Palo Alto Research Center report SSL-79-3.

[6] S. M. Casner. A task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics*, 10(2):111–151, Apr. 1991.

[7] J. Christensen, J. Marks, and S. Shieber. Algorithms for cartographic label placement. In *Proceedings of the American Congress on Surveying and Mapping 1*, pages 75–89, 1993.

[8] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, July 1995.

[9] T. Comver and J. Maltby. Evaluating usability of screen design with layout complexity. In *Proceedings of OZCHI'95, the CHISIG Annual Conference on Human-Computer Interaction*, Full Papers, pages 175–178, 1995.

[10] G. Di Battista, W.-P. Liu, and I. Rival. Bipartite graphs, upward drawings, and planarity. *Information Processing Letters*, 36(6):317–322, Dec. 1990.

[11] S. Feiner. A grid-based approach to automating display layout. In *Proceedings of Graphics Interface '88*, pages 192–197, June 1988.

[12] S. Feiner, J. Mackinlay, and J. Marks. Automating the design of effective graphics: Tutorial notes. AAAI '93, Washington, DC, July 11–16, 1993.

[13] Graph Drawing Conference. http://www.cs.virginia.edu/ gd2000, 2000.

[14] W. Graf. Constraint-based graphical layout of multimodal presentations. In T. Catarci, M. F. Costabile, and S. Levialdi, editors, *Advanced Visual Interfaces, AVI*, volume 36 of *Series in Computer Science*, pages 365–385. World Scientific, 27–29 May 1992.

[15] M. Hofri. Two-dimensional packing: Expected performance of simple level algorithms. *Information and Control*, 45:1–17, 1980.

[16] B. K. P. Horn. *Robot Vision*. MIT Press, Cambridge, MA, USA, 1986.

[17] T. C. Hu and E. S. Kuh. *VLSI Circuit Layout: Theory and Design*. IEEE, USA, 1985.

[18] S. Hudson and I. Smith. subartic ui toolkit user manual. http://www.cc.gatech.edu/gvu/ui/sub_arctic/sub_arctic/doc /users_manual.html.

[19] S. E. Hudson and S. P. Mohamed. Interactive specification of flexible user interface displays. *ACM Transactions on Information Systems*, 8(3):269–288, July 1990.

[20] S. E. Hudson and I. Smith. Ultra-lightweight constraints. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 147–155, 1996.

[21] A. Hurlburt. *The Grid*. Van Nostrand Reinhold Company, Melborne, Australia, 1978.

[22] R. Jeffries, J. R. Miller, C. Wharton, and K. M. Uyeda. User interface evaluation in the real world: A comparison of four techniques. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, Practical Design Methods, pages 119–124, 1991.

[23] Java foundation classes: Now and the future. Whitepaper, http://java.sun.com/products/jfc/whitepaper.html.

[24] R. Kamimura. Application of temporal supervised learning algorithm to generation of natural language. In *International Joint Conference on Neural Networks, Washington DC, January 1990*, volume 1, pages 201–208, Piscataway, NJ, 1990. IEEE.

[25] W. C. Kim and J. D. Foley. DON: user interface presentation design assistant. In ACM, editor, *UIST. Third Annual Symposium on User Interface Software and Technology. Proceedings of the ACM SIGGRAPH Symposium, Snowbird, Utah, USA, October 3–5, 1990*, pages 10–20, New York, NY 10036, USA, Oct. 1990. ACM Press.

[26] S. Kochhar, J. Marks, and M. Friedell. Interaction paradigms for human-computer cooperation in graphical-object modeling. In *Proceedings of Graphics Interface '91*, pages 180–191, June 1991.

[27] C. Kosak, J. Marks, and S. Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1993. To appear.

[28] D. Kurlander and S. Feiner. Inferring constraints from multiple snapshots. *ACM Transactions on Graphics*, 12(4):277–304, Oct. 1993.

[29] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. B.G. Teubner, Stuttgart, 1990.

[30] M. A. Linton, J. M. Vlissides, and P. R. Calder. Composing user interfaces with InterViews. *IEEE Computer*, 22(2), Febuary 1989.

[31] P. Lüders, R. Ernst, and S. Stille. An approach to automatic display layout using combinatorial optimization algorithms. *Software Practice and Experience*, 25(11):1183–1202, Nov. 1995.

[32] I. S. MacKenzie and W. Buxton. Extending Fitts' law to two-dimensional tasks. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, Perception/Performance Theory for HCI, pages 219–226, 1992.

[33] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, Apr. 1986.

[34] A. K. Mackworth. Constraint satisfaction. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1. Addison-Wesley Publishing Company, 1992. Second Edition.

[35] A. K. Mackworth. The logic of constraint satisfaction. *Artificial Intelligence*, 58(1–3):3–20, Dec. 1992.

[36] T. Masui. Evolutionary learning of graph layout constraints from examples. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Demonstrational User Interfaces, pages 103–108, 1994.

[37] J. McCormack, P. Asente, R. Swick, and D. Converse. *X Toolkit Intrinsics—C Language Interface*. Digital Equipment Corporation, Maynard, MA, USA, 1985.

[38] Microsoft Corp. *Microsoft Visual C++ MFC Library Reference*. Microsoft Press, Redmond, WA, USA, 1997.

[39] V. J. Milenkovic, K. M. Daniels, and Z. Li. Automatic marker making. In T. Shermer, editor, *Proceedings of the Third Canadian Conference on Computational Geometry*, pages 243–246, August 6–10 1991.

[40] J. Müller-Brockmann. *Grid Systems in Graphic Design*. Arthur Niggli Publishers, Niederteufen, Switzerland, 1981.

[41] B. A. Myers, R. G. McDaniel, and D. S. Kosbie. Marquise: Creating complete user interfaces by demonstration. In *INTERCHI '93, Human Factors in Computing Systems*, Apr. 1993.

[42] B. A. Myers et al. The garnet user interface development environment. In *Proceedings of the CHI '94 conference companion on Human factors in computing systems*, pages 457–458, 1994.

[43] N. J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, San Francisco, 1998.

[44] J. K. Ousterhout. *Tcl and Tk Toolkit*. Addison-Wesley, 1994.

[45] S. Pan and K. McKeown. Learning intonation rules for concept to speech generation. In *Proc. of the Joint International Conference on Computational Linguistics and Association for Computational Linguistics (COLING-ACL)*, 1998.

[46] A. Pizano, Y. Shirota, and A. Iizawa. Automatic generation of graphical user interfaces for interactive database applications. In B. Bhargava, T. Finin, and Y. Yesha, editors, *Proceedings of the 2nd International Conference on Information and Knowledge Management*, pages 344–355, New York, NY, USA, Nov. 1993. ACM Press.

[47] S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1 of *Active Support for Interaction*, pages 112–117, 1994.

[48] S. F. Roth, J. Mattis, and X. Mesnard. Graphics and natural language as components of automatic explanation. In J. Sullivan and S. Tyler, editors, *Intelligent User Interfaces*. Addison-Wesley, Reading, MA, 1991.

[49] S. Salzberg, D. Searls, and S. Kasif. *Computational Methods in Molecular Biology*. Elsevier Science Ltd., 1998.

[50] E. Schlungbaum. Individual user interfaces and model-based user interface software tools. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces*, pages 229–232, 1997.

[51] A. Sears. Layout appropriateness: A metric for evaluating user interface widget layout. *IEEE Transactions on Software Engineering*, 19(7):707–719, July 1993.

[52] A. Sears. AIDE: A step toward metric-based interface development tools. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Evaluation, pages 101–110, 1995.

[53] A. Sears and A. M. Lund. Creating effective user interfaces. *IEEE Software*, 14(4):21–24, July / Aug. 1997.

[54] F. Shahrokhi, L. A. Szekely, O. Sykora, and I. Vrt'o. Drawing graphs on surfaces with few crossings. *Algorithmica*, 16(1):118–131, July 1996.

[55] G. Singh and M. Green. Automating the lexical and syntactic design of graphical user interfaces: The uofA* UIMS. *ACM Transactions on Graphics*, 10(3):213–254, 1991.

[56] G. Singh, C. H. Kok, and T. Y. Ngan. Druid: A system for demonstrational rapid user interface development. In *Proc. of the 3rd Annual Symposium on User Interface Software and Technology (UIST'90)*, pages 167–177, Snowbird, UT, 1990.

[57] C. S.-M. Tim Bray, Jean Paoli and E. Maler. Extensible markup language (xml) 1.0. http://www.w3.org/TR/2000/REC-xml-20001006.

[58] T. S. Tullis. A computer-based tool for evaluating alphanumeric displays. In *Proceedings of IFIP INTERACT'84: Human-Computer Interaction*, Evaluation—Approaches and Methods, pages 719–723, 1984.

[59] B. Vander Zanden and B. A. Myers. Automatic, look-and-feel independent dialog creation for graphical user interfaces. In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, Constraint Based UI Tools, pages 27–34, 1990.

[60] L. Weitzman and K. Wittenburg. Relational grammars for interactive design. In E. P. Glinert and K. A. Olsen, editors, *Proc. IEEE Symp. Visual Languages, VL*, pages 4–11. IEEE Press, 24–27 Aug. 1993.

[61] L. Weitzman and K. Wittenburg. Automatic presentation of multimedia documents using relational grammars. In *Proceedings of the Second ACM International Conference on Multimedia (MULTIMEDIA '94)*, pages 443–452, New York, Oct. 1994. ACM Press.

[62] M. X. Zhou and S. K. Feiner. Data characterization for automatically visualizing heterogeneous information. In *Proceedings IEEE Symposium on Information Visualization*, pages 13–20. IEEE, 1996.

[63] M. X. Zhou and S. K. Feiner. Top-down hierarchical planning of coherent visual discourse. In J. Moore, E. Edmonds, and A. Puerta, editors, *Proceedings of the International Conference on Intelligent User Interfaces*, pages 129–136, New York, 1997. ACM Press.

[64] M. X. Zhou and S. Ma. Toward applying machine learning for to design rule acquisition for automated graphics generation. Technical report, IBM Watson Research Center, 1999.