



Active Layout Engine: Algorithms and Applications in Variable Data Printing

Xiaofan Lin
Digital Printing and Imaging Laboratory
HP Laboratories Palo Alto
HPL-2005-6(R.1)
December 8, 2005*

automatic
document layout,
constraint solving,
table formatting,
Simplex, variable
data printing

Variable Data Printing (VDP) provides a huge opportunity for HP's imaging and printing business. A core technology required by highly customized VDP applications is the automatic layout engine, whose task is to adjust the original design or generate a new layout to present variable contents. A brand-new layout engine, called Active Layout Engine (ALE), has been designed and implemented. "Active" reflects several unique features of the engine: First, through (multi)linear text block modeling and two-pass constraint solving algorithm, it supports a rich set of layout operations, such as simultaneous optimization of text block width and height, integrated image cropping, and non-rectangular text wrapping. Second, it does not rely on a particular layout description language and thus can actively pursue emerging formats and standards. This report describes the various technical aspects of ALE: Linearization of the text block modeling, two-pass constraint solving algorithm, format-neutral Active Layout Template (ALT), system optimization, and typical VDP applications around the core engine.

Active Layout Engine: Algorithms and Applications in Variable Data Printing

Xiaofan Lin

Hewlett-Packard Laboratories

1501 Page Mill Rd MS 1203, Palo Alto, CA 94304

Email: xiaofan.lin@hp.com

Abstract

Variable Data Printing (VDP) refers to the process of generating and printing dynamic or personalized contents. A core technology required by highly customized VDP applications is the automatic document layout design engine, whose task is to adjust the original design or generate a new layout to present variable contents. This paper presents a novel document layout design engine, called Active Layout Engine (ALE). “Active” reflects several unique features of the engine: First, through linear text block modeling and two-pass constraint solving algorithm, it supports a rich set of layout operations, such as simultaneous optimization of text block width and height, integrated image cropping, and non-rectangular text wrapping. Second, it does not rely on a particular layout description language and thus can actively pursue emerging formats and standards. This paper describes the various technical aspects of ALE: linearization of the text block modeling, two-pass constraint solving algorithm, format-neutral Active Layout Template (ALT), system optimization, and typical VDP applications around the core engine.

Key words: automatic document layout, constraint solving, table formatting, Simplex, variable data printing

1. Introduction

Automatic layout and composition technology is of great value to end-to-end digital publishing solutions because it can relieve or eliminate the bottleneck of creating documents composed of highly customized text and image contents. It is also a very challenging technical problem since it involves 2-D optimization of positions and dimensions of multiple types of contents: images, texts, and vector graphics. Thus, there has been extensive research in this area. One of earliest efforts may be attributed to the Juno-2 constraint-based drawing editor, developed by Heydon and Nelson in the early 1990's [1]. Badros et al [2] proposed a constraint extension to Scalable Vector Graphics (SVG) to enable interactive graphics on the Web. Jacobs et al [3] introduced an adaptive document layout system that automatically selects the best template for given contents. Purvis et al [4] formalized the creation of personalized documents as a multi-objective optimization problem and used a genetic algorithm to automatically assemble such documents. Johari et al [5] created a specialized pagination and layout system for yellow pages. Berkner et al [6] introduced a method to intelligently scale picture and text portions of an image by utilizing information available in the JPEG2000 file. Atkins [7] proposed an image layout algorithm to automate the design of photo albums. Agrawala [8] described a heuristic search-based layout algorithm for placing labels on maps to improve the usability of route maps. In addition, many placement and routing algorithms developed for electronic and mechanic CAD can be also leveraged for document layout design.

Obviously, automatic document layout is a very wide area and no single algorithm can solve all of the challenges. Figure 1 shows the taxonomy of various document layout algorithms in terms of template flexibility and content flexibility. The left side corresponds to the fixed copy

hole fitting, in which variable contents (for example, names, addresses, etc.) are simply filled into fixed copy holes where each object has predefined size and location. On the far right are template-free approaches, in which there are no predefined relative positions among the objects. In the middle, existing templates are needed to define the relative positions among the objects, but they are allowed substantial flexibility with regard to the absolute positions and dimensions of the individual objects. Another criterion of the taxonomy is the content flexibility. At the very bottom, the contents are restrained to only pure image/text/graphic type. In the middle, different types of contents are allowed but they are kept largely static throughout the layout algorithms (for example, the text block width and the image aspect ratio are fixed). On the top, the contents can actively morph in the layout process to achieve the best overall layout. For example, the text blocks can take different widths, the images can be cropped to various aspect ratios, and the vector graphics can scale within a certain range. One possibility of content adaptation is to select the appropriate image and text contents in order to achieve good layouts [3]. As shown in Figure 1, Active Layout Engine (ALE), the subject of this paper, depends on existing flexible templates, supports mixed types of contents, and allows a very high degree of content flexibility (mixed contents, variable aspect ratios of text and image blocks, non-rectangular text wrapping, etc.).

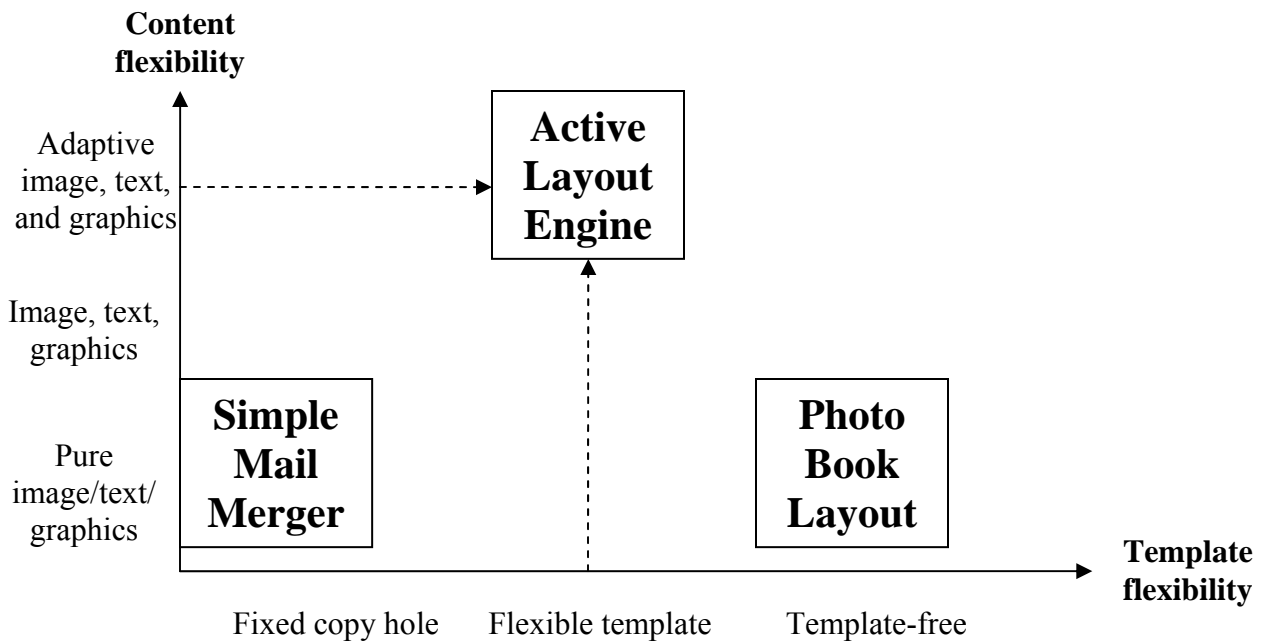


Figure 1: Taxonomy of document layout algorithms based on template and content flexibility

ALE systematically calculates out the layout through mathematical optimization instead of trying out the layout. Along this direction, constraint solving is adopted. On the other hand, conventional constraint solving methods such as Simplex can only handle linear constraints. So a number of key innovations are introduced into ALE:

- **Linear text block modeling** enables the use of Simplex rather than other less efficient nonlinear constraint solving methods.
- **Two-pass constraint solving algorithm** effectively compensates for the inaccuracy introduced by linearization.
- **Active Layout Template (ALT)** provides a format-neural schema of describing adjustable layouts.

Section 2, 3, and 4 discuss the three key techniques in detail. Section 5 presents several applications of ALE: document versioning, template-specific new layout generation, table formatting, and template-free catalog creation. Section 6 is devoted to the system optimization. Section 7 summarizes the paper and points out future research directions.

2. Linear text block modeling

Different from many previous methods, we want to allow the individual text blocks to have variable widths in order to obtain the optimal layouts. On the other hand, such freedom in text block width poses a difficult technical challenge. When the text content in a rectangular block is fixed, the block's width (w) and height (h) roughly follow $w*h=a$ (a is a constant), which is a nonlinear relationship between w and h . The exact relationship is even more complex. h is not a continuous function of w and it instead follows a stepwise pattern, as shown in Figure 2.

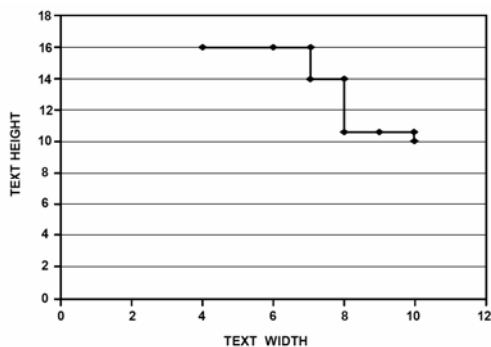


Figure 2: Non-linear relationship between the width and height of a rectangular text block, given the text content

This nonlinear relationship renders many efficient constraint solving methods such as Simplex useless. So our first strategy is to build linear models for the text block height-width

relationship. Based on the targeted applications, there are two cases. For minor layout adjustment, we can use a single linear model to approximate the height-width function of a text block. For more dramatic layout modification or new layout generation, we have to use a cluster of linear models for each text block.

2.1. Use a single linear model

Let us assume the original width of a text block is W_0 and the original height is H_0 . Then we attempt another width W_1 . By accessing the line-breaking function of the rendering engine (for example, FOP [15], an open source software package that can parse and render XSL-FO documents, is used in the current implementation), we can get the new height H_1 . In this way we have a linear approximation of the relationship between the width and height:

$$H = H_0 + k * (W - W_0) \quad (1)$$

$$\text{where } k = (H_1 - H_0) / (W_1 - W_0)$$

2.2. Use multiple linear models

In minor layout adjustment, we can use a single linear function to approximate the width-height function around the original layout. The situation is quite different in generating completely new layouts. Because we do not have any knowledge on the widths of the text blocks, using one linear function will result in intolerable errors (see Figure 3).

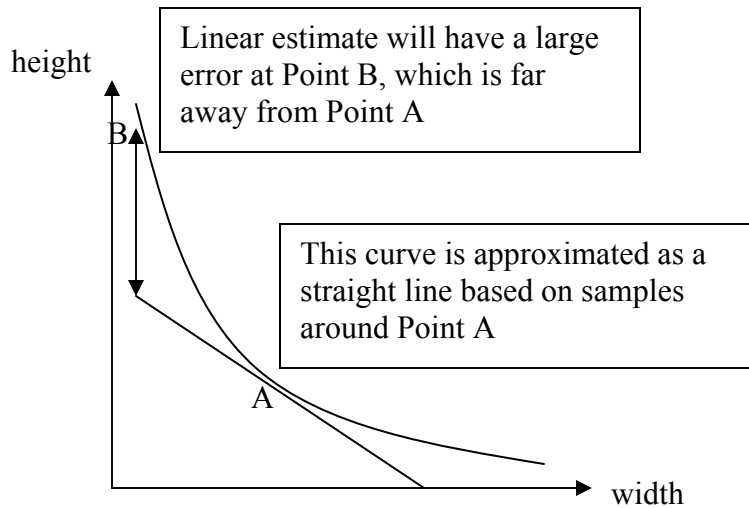


Figure 3: Using one linear segment to approximate the text block height-width relationship can result in large errors

In order to reduce estimation errors, we use a cluster of linear functions to approximate the original nonlinear function. The following several steps are involved:

- Render the text content at several different widths w_1, w_2, \dots, w_n and get corresponding heights h_1, h_2, \dots, h_n respectively. In the current implementation, five (w, h) combinations are sufficient.
- Fit the data points $(w_1, h_1), (w_2, h_2), \dots, (w_n, h_n)$ to a hyperbolic function $h(w) = k/w + b$ (k and b are constant for given text content and format). k and b can be calculated using statistical regression method to achieve the least mean square error (LMS).
- Locate a number of sampling points on $h(w) = k/w + b$ across the maximal allowed range of w . For example, we can find twenty sampling points with w in [50 points, 500 points]. It is also preferred that the intervals between the heights of the sampling points are constant.
- Append a number of linear constraints for each text block to the existing constraints:

$$height \geq h[i] + (h[i] - h[i-1]) * (width - w[i-1]) / (w[i] - w[i-1]), i=1, \dots, 19 \quad (2)$$

Figure 4 shows how a cluster of linear functions are used to approximate the hyperbolic function. It can be seen that if Equation 2 is satisfied for all of the nineteen linear functions, the original nonlinear constraint $height > k/width + b$ will be fulfilled at a very high precision. It is worth noting that the convex nature of the hyperbolic function plays an important role. Otherwise we cannot approximate the nonlinear constraint with a cluster of linear constraints. This is also why we first model the original nonlinear function as a hyperbolic function using regression and then sample the hyperbolic function to get the linear constraints instead of directly sampling the original nonlinear function.

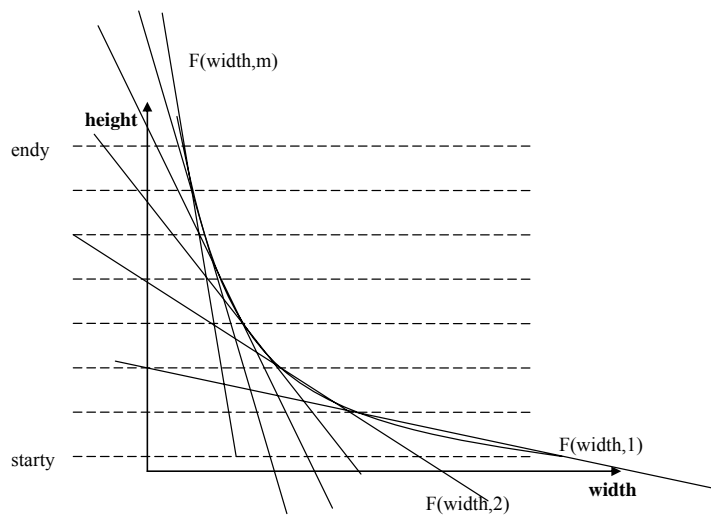


Figure 4: Create a cluster of linear functions to cover the hyperbolic function

One common text layout feature in professional designs is hyphenation. Both the single linear model and the multi-linear model can be applied when hyphenation is used by the text rendering engine. Of course, the parameters k and b will be slightly different from those without hyphenation.

3. Two-pass constraint solving

The linear models are just estimates and cannot guarantee completely correct layouts. Thus, we run two passes of constraint solving. Using the linear models of the text blocks, the first pass decides the optimal width for each text block. Then through actual line-breaking, we can calculate the exact height for each text block. In the second pass, we fix the dimension of each text block and decide the final positions of the text blocks as well as the positions and sizes of the image blocks. Figure 5 shows the workflow of the ALE.

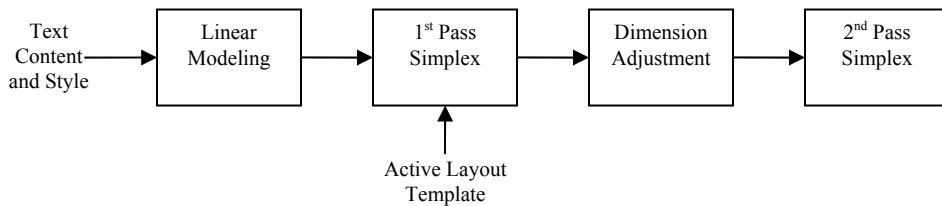


Figure 5: Workflow of Active Layout Engine

The following example illustrates why the two-pass algorithm is necessary and how it works. The left side is the document containing the original contents. Then more text contents have been added to the two text blocks. The layout after the first pass is shown in the middle. As can be seen, this layout is roughly good with Block A expanded horizontally and Block B expanded vertically. The only problem is that the bottom of Block A overlaps with the title text under it. This is due to the approximate nature of the linear models used in the first pass. After the second pass, the overlap problem has been corrected and the bottom image is cropped a little to make the extra room for the text content. In comparison, Figure 7 shows the result if the width of each text block is fixed. The text in Block B runs out of space and is forced to occlude part of the image at the bottom.

Layout with original content

highlights of bora bora

Join us for a brief introduction to Bora Bora. You'll depart from the village of Vaitape on one of the local sightseeing vehicles, "La Truck," for a drive destined to be a highlight of your cruise.


WHAT YOU'LL SEE ALONG THE WAY

A 22-mile road circles the island, taking you past groves of coconut palms and tiny villages hugging the shoreline. You'll marvel at the beauty of Farepiti, Fannui and Anau with their mysterious open-air maases, temples of ancient times. As you travel on, passing Paooa Point and Matira Point, nothing will disturb your enjoyment of this earthly paradise.

TOUR BOB-A


DURATION
Approximately 2 hours

PRICE
\$36 Adult \$21 Child



circle island and hotel marara

Combine the pleasures of an island tour with a visit to the lovely Hotel Marara for sunning on the white sand beaches, native entertainment and more.




WHAT YOU'LL VISIT

Hotel Marara. An exotic, authentically Polynesian retreat, the Hotel Marara is where you will be greeted "Tahitian-style" with a flower couronne, followed by a welcome drink, a light fresh-fruit buffet and an exciting Polynesian show.


TOUR BOB-B

DURATION
Approximately 4 hours

PRICE
\$77 Adult \$48 Child



NOTE:
Time does not permit to both swim and see the show during the visit at Hotel Marara.



Text block width is increased

Overlap due to estimate error of the linear model

Text block height is increased

Overlap is fixed

highlights of bora bora

Join us for a brief introduction to Bora Bora. You'll depart from the village of Vaitape on one of the local sightseeing vehicles, "La Truck," for a drive destined to be a highlight of your cruise.


WHAT YOU'LL SEE ALONG THE WAY

A 22-mile road circles the island, taking you past groves of coconut palms and tiny villages hugging the shoreline. You'll marvel at the beauty of Farepiti, Fannui and Anau with their mysterious open-air maases, temples of ancient times. As you travel on, passing Paooa Point and Matira Point, nothing will disturb your enjoyment of this earthly paradise. Fannui and Anau with their mysterious open-air maases, temples of ancient times. As you travel on, passing Paooa Point and Matira Point, nothing will disturb your enjoyment of this earthly paradise.

TOUR BOB-A


DURATION
Approximately 2 hours

PRICE
\$36 Adult \$21 Child



circle island and hotel marara

Combine the pleasures of an island tour with a visit to the lovely Hotel Marara for sunning on the white sand beaches, native entertainment and more.




WHAT YOU'LL VISIT

Hotel Marara. An exotic, authentically Polynesian retreat, the Hotel Marara is where you will be greeted "Tahitian-style" with a flower couronne, followed by a welcome drink, a light fresh-fruit buffet and an exciting Polynesian show. An exotic, authentically Polynesian retreat, the Hotel Marara is where you will be greeted "Tahitian-style."


TOUR BOB-B

DURATION
Approximately 4 hours

PRICE
\$77 Adult \$48 Child



NOTE:
Time does not permit to both swim and see the show during the visit at Hotel Marara.



After the first pass

highlights of bora bora

Join us for a brief introduction to Bora Bora. You'll depart from the village of Vaitape on one of the local sightseeing vehicles, "La Truck," for a drive destined to be a highlight of your cruise.


WHAT YOU'LL SEE ALONG THE WAY

A 22-mile road circles the island, taking you past groves of coconut palms and tiny villages hugging the shoreline. You'll marvel at the beauty of Farepiti, Fannui and Anau with their mysterious open-air maases, temples of ancient times. As you travel on, passing Paooa Point and Matira Point, nothing will disturb your enjoyment of this earthly paradise.

TOUR BOB-A


DURATION
Approximately 2 hours

PRICE
\$36 Adult \$21 Child



circle island and hotel marara

Combine the pleasures of an island tour with a visit to the lovely Hotel Marara for sunning on the white sand beaches, native entertainment and more.




WHAT YOU'LL VISIT

Hotel Marara. An exotic, authentically Polynesian retreat, the Hotel Marara is where you will be greeted "Tahitian-style" with a flower couronne, followed by a welcome drink, a light fresh-fruit buffet and an exciting Polynesian show. An exotic, authentically Polynesian retreat, the Hotel Marara is where you will be greeted "Tahitian-style."


TOUR BOB-B

DURATION
Approximately 4 hours

PRICE
\$77 Adult \$48 Child



NOTE:
Time does not permit to both swim and see the show during the visit at Hotel Marara.



After the second pass

Image cropped to make extra room for the text

Figure 6: Example of the two-pass constraint solving algorithm

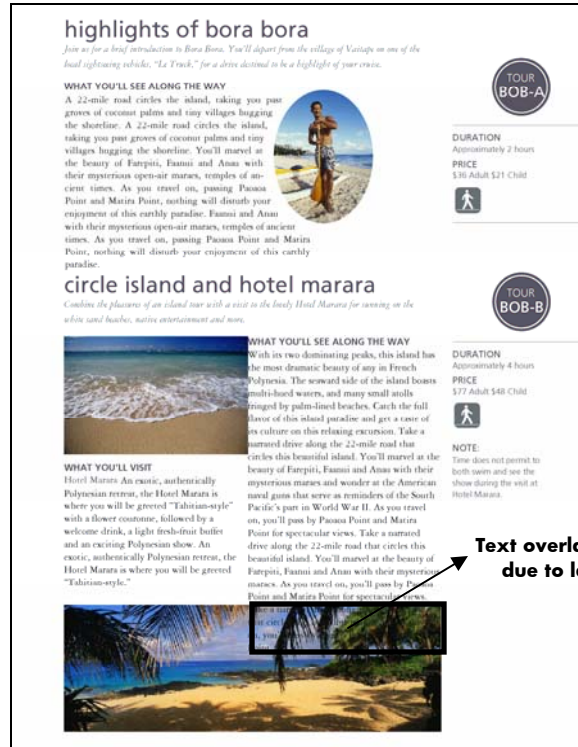


Figure 7: Result without adjusting text block widths

3.1. Constraint solver

We have chosen Cassowary solver [9] because of its several attractive features. First, it supports non-required constraints, such as “strong” and “weak” constraints, in addition to the conventional required constraints. This feature is very convenient in auto-layout systems. In layout adjustment, we can formulate “weak” constraints saying that the position of each object after adjustment should be the same as the original position, and “strong” constraints saying that the inter-block white space should keep the original value. Internally, Cassowary solver will convert the non-required constraints into linear cost functions using quasi-linear techniques. Second, it supports efficient incremental constraint solving. Although we do not utilize this

feature now, it can be very useful to increase the processing speed of high-volume VDP applications by generating a new layout from the existing states of the solver rather than from scratch. Third, Cassowary solver has demonstrated impressive robustness and stability throughout our experiments.

3.2. Constraint formulation

The constraints are formulated in the format adopted by CSVG [2]. They are easy-to-understand infix expressions of equalities or inequalities. There are two parts of constraints in the first pass (see Figure 8(a)). One part is the original constraints from upstream applications. In the layout adjustment shown in Figure 6, the following constraints may be formulated by the constraint inference engine described in our previous paper [10]:

- 1) `<constraint rule="D_left >= A_right + 10" strength="required"/>`
- 2) `<constraint rule="C_top >= A_bottom+ 5" strength="required"/>`
- 3) `<constraint rule="C_top >= D_bottom+ 5" strength="required"/>`
- 4) `<constraint rule="A_right = 120" strength="strong"/>`
-

The first constraint puts the image block D to the right of the text block A by a margin of at least 10. The second and the third constraints make sure that the title block C is under both the text Block A and the image Block D by a margin of at least 5. The three constraints together allow the text block A to change height and/or width to accommodate new content while keeping the original relative positions between the blocks. The fourth one is a “strong” constraint, saying that the right edge of the text block A is preferred to be 120, the value in the original document. Such non-required constraints are created to minimize the change in the layout after adjustment.

The other part is the approximate linear constraints based on the linear text block modeling. Figure 8(b) shows the solution after the first pass. Then we make corrections to the first pass results. As shown in Figure 8(b), based on the linear model, when the Δ width (here Δ means relative value compared with that of the original layout) of P0T5 (P0T5_deltawidth) is 48.71, the height of P0T5 (P0T5_height) is 123.23. On the other hand, we can actually place the text into P0T5 under the condition that P0T5_deltawidth is 48.71, and the height is calculated to be 132, which is different from the linear model. So we use height obtained from actual content placement to replace that based on linear model and generate the constraints for the second pass (see Figure 8(c)). The constraint solver will produce the final solution.

Original constraints:
 <constraint rule="P0I0_left >= P0T5_left" strength="required"/>...
Approximate linear constraints:
 <constraint rule="P0T5_height = 189.00-P0T5_deltawidth*1.3500" strength="required"/>...

(a) Constraints for the first pass

Text block dimensions:
 P0T5_deltawidth = 48.72
 P0T5_height = 123.23...

(b) Results for the first pass

Original constraints are the same as those in the first pass:
 <constraint rule="P0I0_left >= P0T5_left" strength="required"/>...
Approximate linear constraints are replaced by fixed values for text block widths and heights:
 <constraint rule="P0T5_deltawidth = 48.72" strength="required"/>
 <constraint rule="P0T5_height = 132.00" strength="required"/>...

(c) Constraints for the second pass

Figure 8: Example of constraint formation throughout the two passes

3.3. Evaluation and discussion on alternative methods

Because the linear models for the text blocks are approximate, the two-pass algorithm does not guarantee the optimal layout with respect to the original layout constraints. Then an important question is how close the layout generated from this method is to the optimal layout. In order to answer this question, we formulate the constraints and objective function for a simple layout shown in Figure 9 and compare the result from the two-pass algorithm with that from brutal-force search. The total width (w) is set to 400 points, the text is in 12-point Times-Roman font, and the paragraph-based line-breaking algorithm designed by Knuth [18] is used to break text into lines. As described in Section 2.2, twenty linear constraints are created for each text block (Because single-linear modeling can be considered as a fast algorithm of multi-linear modeling, we only evaluate the accuracy of multi-linear modeling). The brutal-force search algorithm tries out a number of densely spaced values for the width of Block 1 within a range centered on the result obtained by the two-pass algorithm and then locate the solution leading to the minimal total height of the layout. We choose this simple layout because the brutal-force search algorithm involves too much computation to be feasible for layouts with more variables.

In order to explore the robustness of our algorithm, we have experimented with different text content combinations (see Table 1). For example, in Case 1, Block 1 contains very long text and the other two blocks have short texts; in Case 2, we have balanced texts across the blocks; in Case 3, the text is very short in every block; and in Case 4, we have much more text in the second column than that in the first column. As shown in Table 1, in three of the four test cases, the two-pass algorithm is able to find an absolute optimal solution that leads to the minimal height. In Case 1, the two-pass algorithm's result is only one text line more than the optimal solution. Figure 10 displays the relationship between the total height (h) and Block 1's width (a).

Because of the discrete nature of line breaking, the optimal height can be achieved in a range of a , which is about [224, 230] in this case. The two-pass algorithm sets a to 226 and the corresponding layout (see Figure 11) is visually balanced on the two columns. If we do not optimize at all and simply divide the total width to two columns of equal width, the total height will be 350 or 10 lines more than the optimal value and the two columns will have very different heights.

We have also made such quantitative comparison on a number of other layouts and have observed similar behavior: The two-pass algorithm can find a solution that is optimal or at least very close to be optimal with regard to the specified cost function. In addition, the visual quality of more complex layouts produced by the two-pass algorithm (see several examples used in this paper) is also within our expectation.

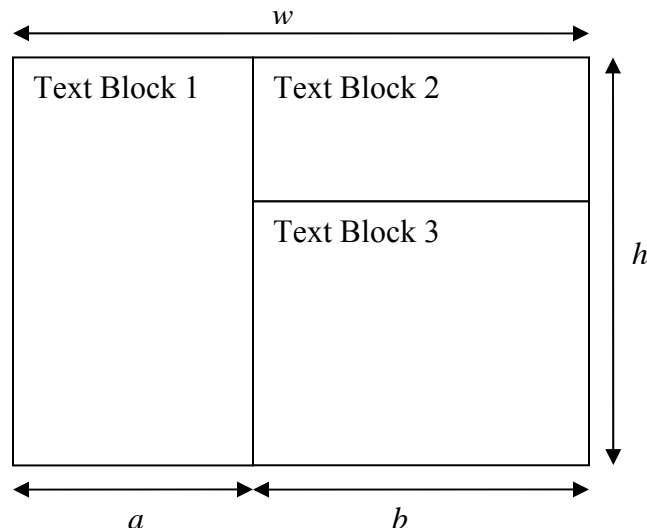


Figure 9: Testing layout problem: One text block is in the first column and two text blocks are in the second column. The total width (w) of the layout is fixed and the objective is to minimize the total height h by changing the first column's width (a).

Table 1: Comparison of the brutal-force search and the two-pass algorithm on different test cases

Case No	Block 1 Length (in chars)	Block 2 Length (in chars)	Block 3 Length (in chars)	Optimal h by brute-force search	h from the two-pass algorithm
1	721	67	269	140	152
2	243	316	230	137	137
3	53	46	67	41	41
4	67	269	721	152	152

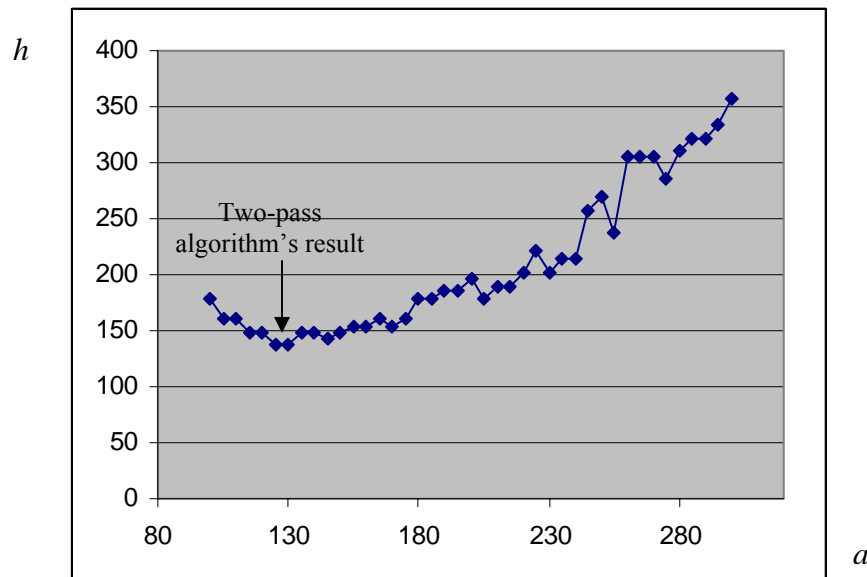


Figure 10: Brutal-force search results on Case 2

<p>2 Two-toned leather and suede thong with whipstitch and perforated detailing. Leather wrapped sole, padded footbed and rubber outsole. Sizes: Whole and half sizes: 5 1/2-10, 11M. Colors: Black, turquoise or white suede/leather combo. \$49.00</p>	<p>4 We're predicting sunny skies and comfortable style all season long. Perfect with skirts and shorts for added fun. Leather slide with a flexible rubber sole and adjustable closure for ease of fit. " heel. Sizes: Whole and half sizes: 5 1/2-10, 11M. Colors: Black, white, red, gold or silver leather. \$49.00</p>
	<p>3 Two-toned suede and leather slide with whipstitch and perforated detailing. Leather wrapped sole, padded and rubber outsole. Sizes: Whole and half sizes: 5 1/2-10, 11M. Colors: Gold, black or pink leather/suede combo.</p>

Figure 11: Layout from the two-pass algorithm on Case 2

In literature there are some efficient algorithms, such as the work described by Wang in [14], along the direction of heuristic search, which can handle more complex layouts. We are not using them as references in this evaluation for several reasons. First, some existing algorithms do not solve exactly the same problem we want to solve. For example, Wang’s methods only find out a feasible solution to satisfy constraints while we want a solution that is also optimal to a certain cost function. Second, many heuristic search algorithms are approximate themselves in order to be efficient while we want to compare the two-pass algorithm with the real ground truth. Third, because there are not publicly available benchmarking platform and data set on this problem, the only way to make comparison is to implement alternative algorithms in-house, which is a challenging task. For example, Mixed Integer Programming (MIP) [19], basically a heuristic search algorithm, can solve the nonlinear constraints shown in Figure 2. We have to get the accurate step-wise height-width function in order to use MIP. That is a hard problem by itself and requires substantial amount of computation.

4. Active Layout Template

Besides the core algorithms, another important aspect is how to describe adjustable layouts. Although many standard formats, such as SVG and XSL-FO [20], are designed for static layouts, few formats are available for adjustable layouts. Badros et al have designed Constraint SVG (CSVG) [2] to support adjustable graphics by introducing variables into standard SVG. We extend this idea to convert almost any XML-based layouts to adjustable templates, called Active Layout Templates (ALT). Working together with constraints, it only provides a description framework for flexible layouts. On the other hand, ALT itself does not address what objects and variables should be included and what constraints are necessary to guarantee good layouts. It is

up to the particular applications to fill in the ALT and constraints. For example, in document versioning to be discussed in Section 5.1, Chao and Lin [10] proposed algorithms to extract the ALT and constraints based on the analysis of the original PDF file. Kurlander and Feiner [11] also attempted to infer the constraints governing the graphic objects based on several snapshots.

In ALT, the numerical values of the attributes in XML layout documents are replaced with variables and expressions (see Figure 12). Special characters are introduced to avoid ambiguities and also to concatenate variables into expressions. In the current implementation, “##” means this attribute value is generated for ALT purpose rather than an ordinary string (for example, “absolute” in the following example). “!!” separates the symbol/expression from the unit (for example, “pt”). “+” is used in expressions to mean that the two sides of “+” should be added together.

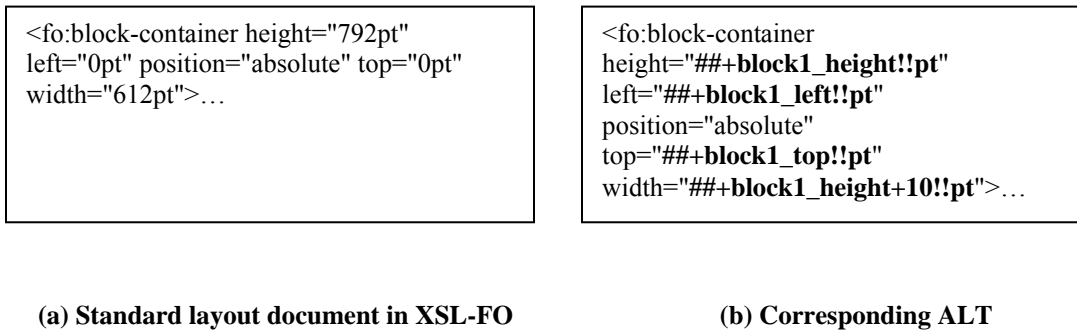


Figure 12: Conversion of standard layout document to ALT

In layout adjustment applications, the ALT is automatically generated on the basis of the original layout document using a converter. The converter parses the original layout XML file and replaces numerical values with expressions and variables according to the protocols introduced earlier. We have also defined a simple variable naming protocol. For example, if a block’s ID is “block1”, then the height will be named “block1_height” and the left coordinate will be named “block1_left”, etc. In layout generation applications, the ALT can be directly

supplied by upstream components, such as the converter in catalog system and the table adaptor in the table formatting system. Section 5 will give more details.

The ultimate goal of building ALT is to generate new documents for new contents. In order to achieve this goal, several steps are needed:

Step 1: Find out the values of the variables based on the new contents (texts, images, etc.). This is accomplished by the two-pass constraint solving algorithm. For the example shown in Figure 12, the layout adjustment algorithm may decide that: block1_height=100, block1_left=10, block1_top=20.

Step 2: Produce standard layout document. The ALT is parsed using DOM or SAX APIs. Each attribute value is analyzed and each expression, identified by the leading “##”, is replaced with the result calculated from the variable values. For the example of Figure 12, the result of this step is

```
<fo:block-container height="100pt" left="10pt" position="absolute" top="20pt"
width="110pt">
```

ALT is a very flexible schema that works on diverse layout formats and it supports many layout operations. The black frames shown in Figure 13 are expressed as an embedded SVG element inside XSL-FO. In order to support vertically expandable frame, a scaling operator is added: <g transform="scale(1,##+Y_factor!!)">. When Y_factor takes different values based on the text content, the frame can then scale accordingly. Similarly, the image cropping shown in Figure 6 is implemented by introducing variables into the SVG clipping path.

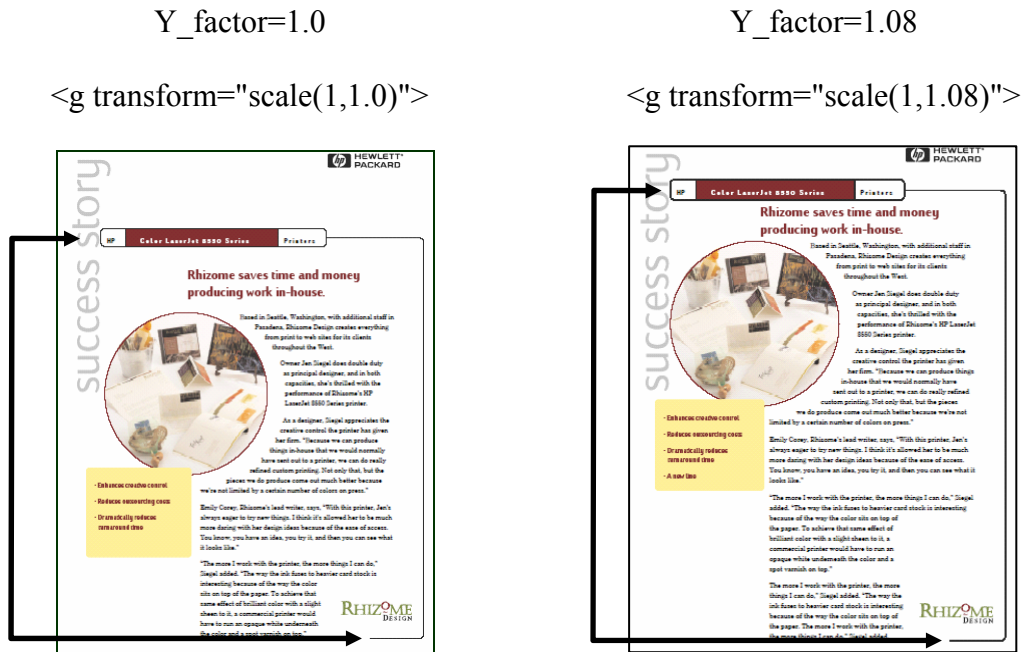


Figure 13: Use ALT to support adjustable SVG elements (The black frames pointed by the arrows are SVG elements controlled by the scaling transforms shown at the top)

5. Applications

ALE has been adopted by a wide spectrum of VDP research prototype applications.

Figure 14 shows the typical scenario of how ALE interacts with other components. The upstream components produce ALT, constraints, and variable contents and feed them into ALE. Inside ALE, the two-pass constraint solving module will provide the Cassowary a set of constraints, including the original constraints and the extra constraints for the text blocks. Using the ALT and values for the variables, we can produce the layout in XSL-FO format, which can be further rendered into PDF. In some applications, such as the catalog generation to be described in Section 5.4, the results from ALE will be fed back to the upstream components to guide further

iterations. The rest of this section describes several prototype applications of ALE: document versioning, template-specific layout generation, table formatting, and catalog creation.

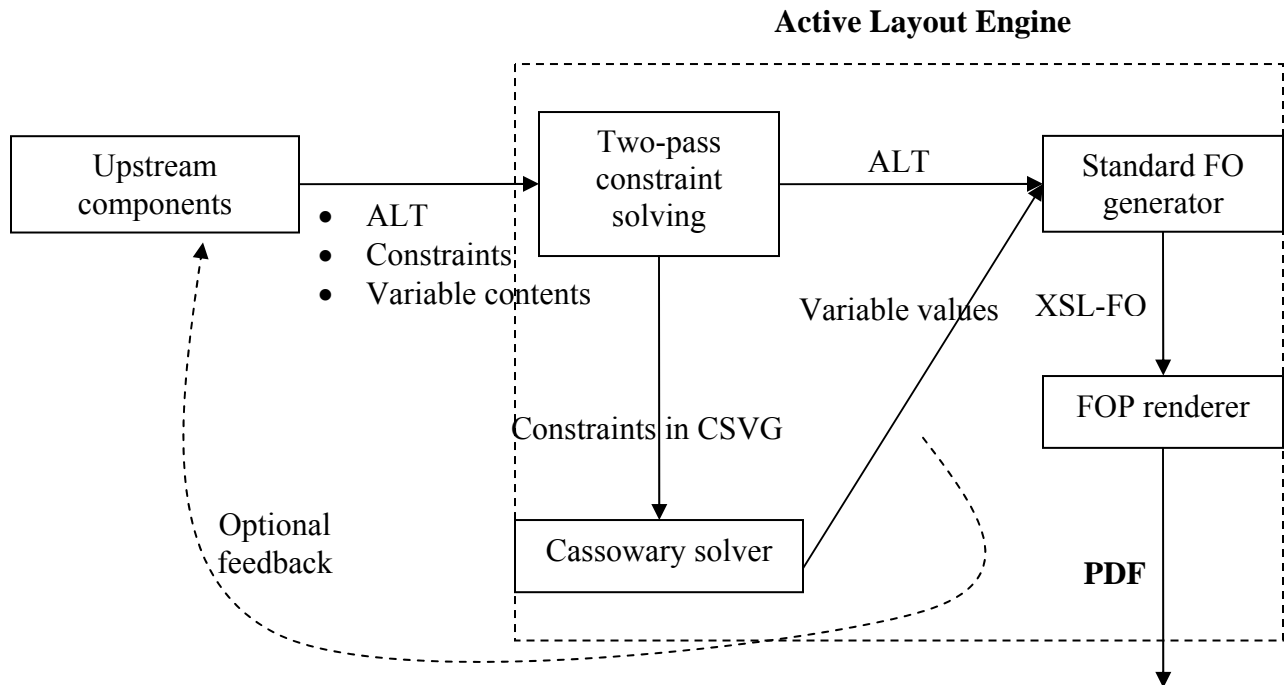


Figure 14: Typical usage pattern of ALE

5.1. Document versioning [13]

In document versioning, ALE works with the layout/constraint extraction component by Chao [10]. Figure 15 shows the system configuration. The layout is extracted from the original PDF as a standard XSL-FO file and other embedded SVG files, which are then transformed into ALT using the converter discussed in Section 4.1. The constraint extractor infers the constraints from the original layout and passes them to ALE. A web-based UI provides new image and text contents. ALE then calculates the new layout and produces the corresponding PDF. In this application, a single linear model is created for each text block because we just make minor adjustments to the layout. Figure 6 and Figure 13 both demonstrate the adjustment examples. As mentioned in the introduction, this is the original application of ALE.

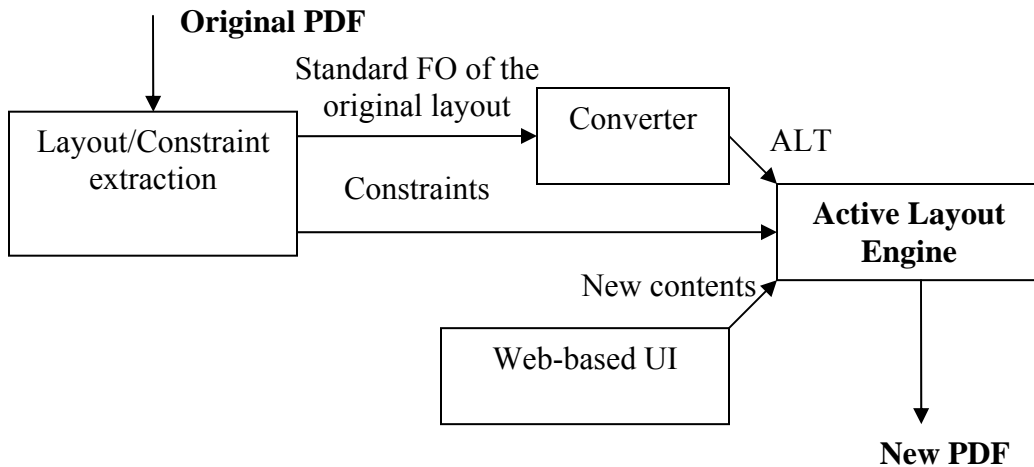


Figure 15: Document versioning system

5.2. Template-specific new layout generation [12]

In contrast to document versioning, layout generation does not start with an existing layout. The upstream applications directly produce ALT and constraint files. Multiple linear models are created for each text block. A good example of this application is the personalized postcard system, which produces postcard layouts (see Figure 16) based on the customer’s selected images and personal information.



Figure 16: Example of template-specific new layout generation

5.3. Table formatting

A common task in publishing is table formatting, whose goal is to decide the optimal dimensions of table cells given the contents and the table’s grid structure. The optimal table layout is usually the one that leads to the minimal table height and satisfies other aesthetic criteria such as balance and symmetry. Formatting a complex table can be a challenge even for humans because the optimal table layout can only be achieved by making the right choice for every cell (see the following sample table, which is cited from Wang’s Ph.D. thesis [14]).

Table 2: A sample table

	x1	x2	x3	x4	x5	x6	x7
y1	Time	Monday	Tuesday	Wednesday	Thursday	Friday	
y2		Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering	
y3	Morning 9:00-12:00	This section is for those who don’t know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to have a career in the software industry in the future.				
y4	Afternoon 1:00-4:00		This section is for those who already know something about computer science and intend to learn how to write simple programs.		This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.		
y5	Evening 7:00-10:00		This section is for those who don’t know anything about computers and intend to learn how to write simple programs.				
y6							

Existing methods of table formatting, such as those proposed by Wang, are various heuristic search methods with worse-case exponential and average/best-case polynomial computational time with reference to the number of cells, columns, and rows. ALE provides an efficient solution to this problem. Figure 17 displays the system configuration. We first describe the table in an XML-based language, which can be converted to ALT and linear constraints. Each cell is defined as a text block in the ALT. As displayed in Table 2, x_1 , x_2 , \dots , and x_n

specify the ordered x-coordinates of the left and right hand sides of the various cells, and y_1 , y_2 , ..., and y_n specify the ordered y-coordinates that a cell can start or end. Inequality constraints will be established to reflect the relative positions of those coordinates. If necessary, extra constraints can be added to satisfy style requirements, for example, making the widths of certain cells the same. Then ALE can solve this text layout problem. Figure 18 demonstrates one composed table.

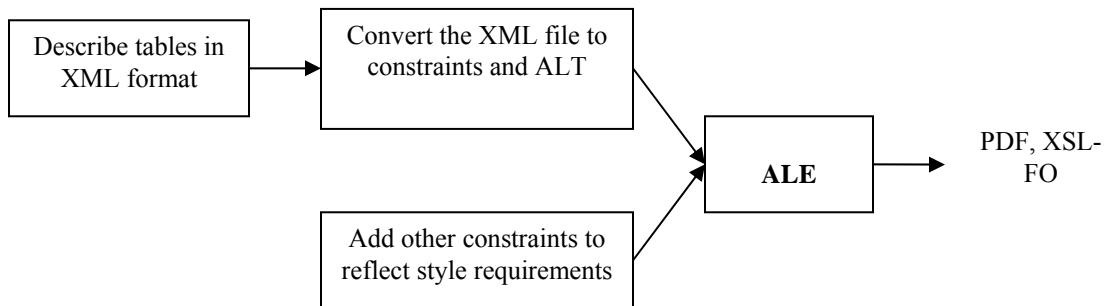


Figure 17: Application of ALE to table formatting

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System software	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to learn how to write simple programs.	This section is for those who already know something about computer science and intend to learn how to write simple programs.	This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.	This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Afternoon 1:00-4:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.	This section is for those who already know something about computer science and intend to learn how to write simple programs.	This section is for those who already know something about computer science and intend to learn how to write simple programs.	This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.	This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.	This section is for those who already know something about computer science and intend to learn how to write simple programs.	This section is for those who already know something about computer science and intend to learn how to write simple programs.	This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.	This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.

Figure 18: Table formatting result using ALE

5.4. Template-free catalog creation

The goal of this application (see Figure 19) is “Content in, layout out”, eliminating handcrafted layout templates. The input to the system is simply pairs of image and associated text descriptions. The system incrementally generates a number of slicing structures using an algorithm originally developed for photo book layout [7]. Since the slicing structure is constructed by sequentially adding one object after another, the final solution is not necessarily the global optima. In fact, other holistic slicing structure generation methods, such as genetic algorithms [4], should also work with ALE to generate catalog layouts. A converter program is created to translate the slicing structure into ALT and constraints, which are sent to ALE to generate the concrete layouts. The best layout will be selected as the final layout. Figure 20 exhibits a couple of layouts from the catalog system.

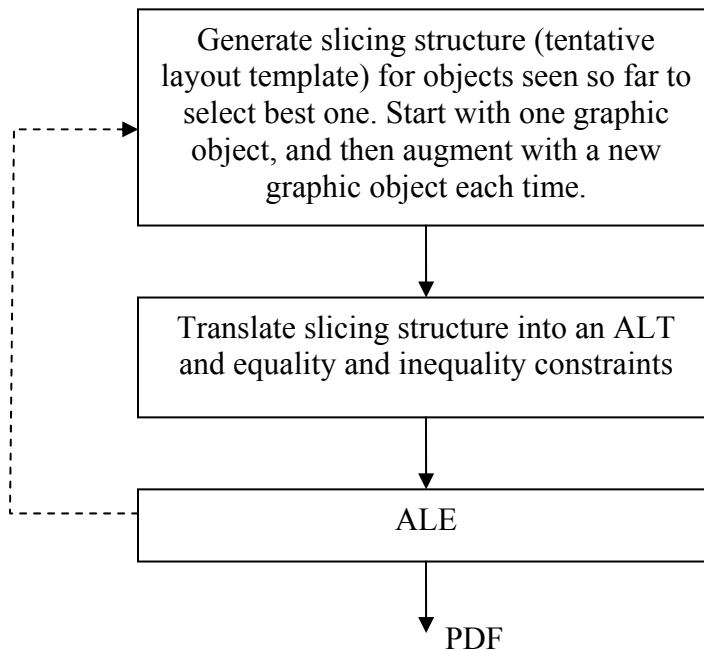


Figure 19: Workflow of the template-free catalog generation system

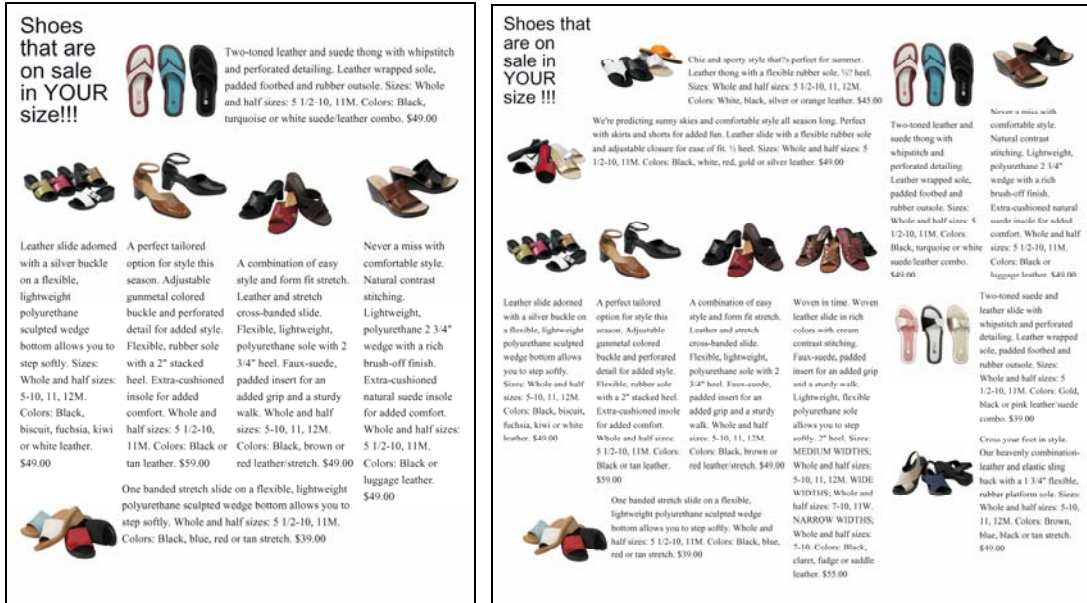


Figure 20: Generated catalog layouts

6. System optimization

The entire ALE is implemented in Java and internally invokes a number of third-party Java packages such as FOP [15] and Cassowary solver [9]. So loading ALE has considerable time overhead. In order to speed up the engine, we have created a client/server mode for ALE (see Figure 21). The heavyweight ALE is started as a server in advance. It then listens to a particular TCP socket for incoming requests and executes the two-pass constraint solving algorithm. The ALE clients can be very lightweight shell programs simply sending requests to ALE server through TCP sockets. In this way, we can avoid the overhead associated with loading the engine repeatedly. In addition, more speedup can be achieved through fine-tuning the parameters of Java Virtual Machine (JVM). Because TCP/IP is a standard protocol supported by different programming languages, this approach also makes ALE accessible to client programs in various languages (C, Java, C#, etc).

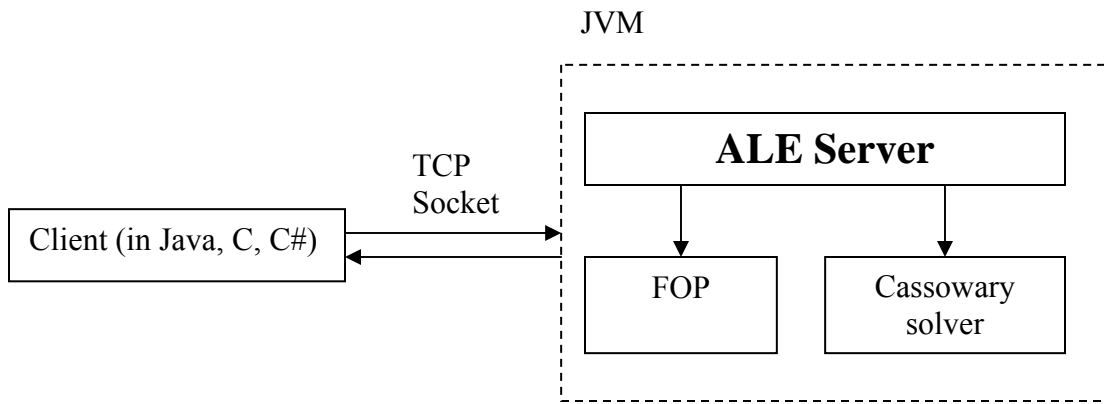


Figure 21: ALE's client/server mode

The following figure demonstrates the effects of the client/server mode. It compares the execution time of the first attempt and the second attempt after the ALE server is started. The task here is to adjust the document shown in Figure 6. The running time is broken down to several major tasks. Text Modeling 1 corresponds to building the linear models for text blocks and creating the constraint files for the first pass. Simplex 1 is the actual first pass Cassowary solver invocation. Text Modeling 2 relates to modifying text block dimensions and generating the constraint files for the second pass. Simplex 2 reflects the second pass Cassowary solver invocation. The second attempt is almost three times as fast as the first attempt because all of the necessary class/JAR files are already loaded and the JIT compiler has optimized the binary code after the first attempt.

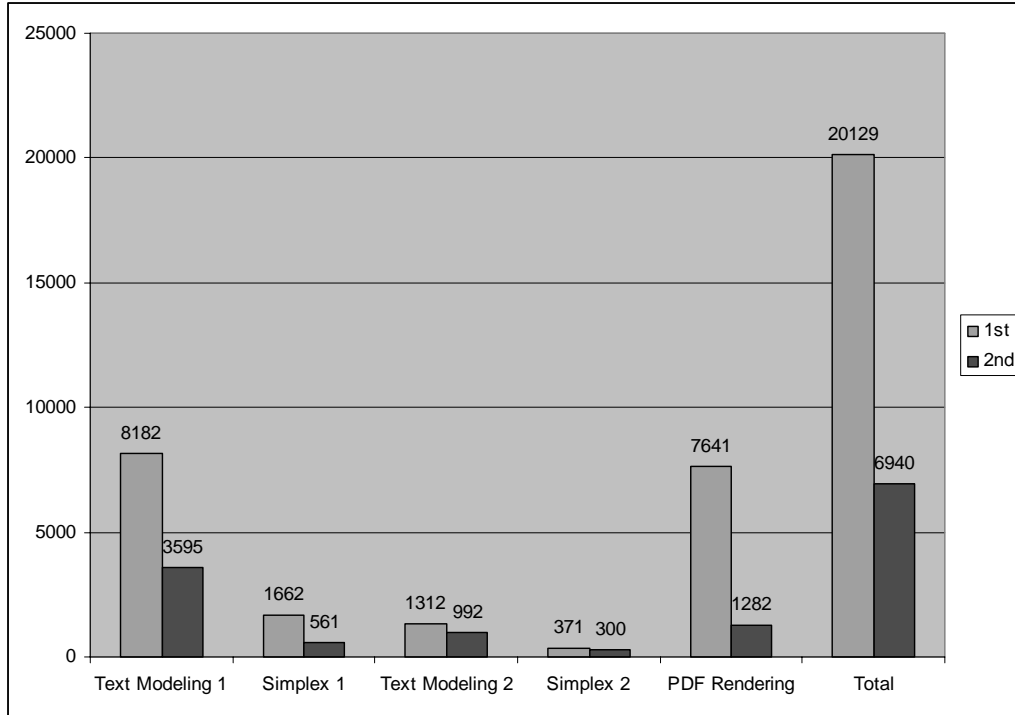


Figure 22: Speedup of client/server mode (y-axis corresponds to processing time in milliseconds)

In addition, the algorithm of ALE can be optimized for different applications. In the catalog creation application, ALE is invoked hundreds of times. The second pass constraint solving can be removed in the intermediate iterations until we have got the final slicing structure. This technique can accelerate the overall process significantly. With all of the above system optimization techniques, the speed of the catalog layout system has been increase by over six times. Figure 23 breaks down the speedup. The time to generate an 11-item catalog page has been reduced to 150 seconds from the original 980 seconds on a 2.8GHz Windows XP computer.

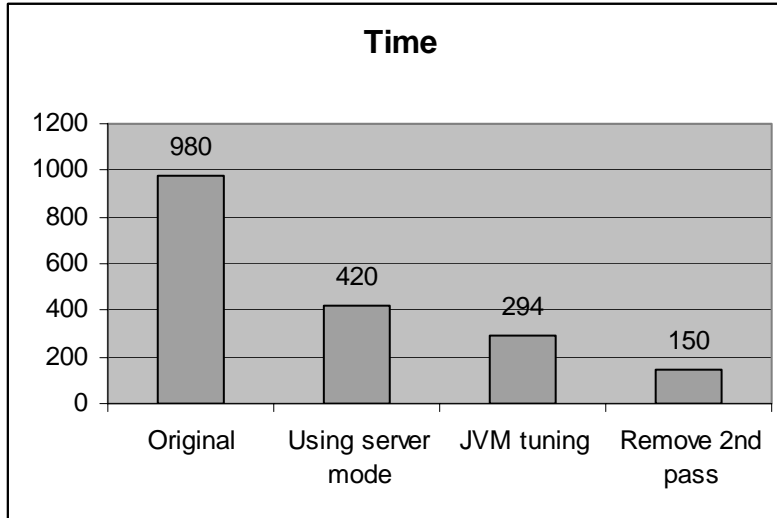


Figure 23: Breakdown of the speedup (y-axis corresponds to processing time in seconds)

7. Conclusions

Through several innovations such as linear text block modeling, two-pass constraint solving algorithm, and Active Layout Template, ALE provides a robust and generic solution to relieve or eliminate the bottleneck on the creation side of automated digital publishing pipeline. Its power has been demonstrated a number of applications. On the other hand, there remain some interesting and valuable future research topics:

In document versioning, although the analysis method described in [10] can automatically extract most of the templates and constraints correctly, it does make mistakes from time to time and then we have to directly edit the ALT and CSVG files to make necessary corrections. Of course, it is not realistic to expect the real-world end users to have such abilities. In order to make this technology more practical, it is essential to create a friendly graphic user interface that allows the interactive modification of the constraints and templates. Besides, ALE is now only addressing how to get the mathematically feasible layouts. The next question is how to automatically formulate constraints that lead to aesthetically appealing layouts [16][17].

As described in Section 3.3, the two-pass algorithm is still an approximate solution because the text block height correction may compromise the original constraints related to text block heights. In practice, the height correction is usually minor, within one or two lines. For example, in the finished layouts shown in Figure 18 and Figure 20, the bottoms of several text blocks are roughly aligned as the result of enforcing the “minimal total height” constraint. Mathematically, the most compact layout can only be achieved when the text blocks have balanced heights. The two-pass algorithm has balanced the different text blocks’ heights even as an approximate solution. If some applications require strict enforcement of height equalities, we can fine-tune word/character spacing to squeeze the text into the height calculated by the first pass. Knuth et al proposed such techniques in TeX [18].

In this paper, we are mostly concerned about constructing document layouts of the same page size to accommodate variable content data. On the other hand, the same techniques can also be used to lay out the same contents in various sizes, such as cell phone, regular monitor, and different paper sizes. This is another interesting area for future research.

Acknowledgements

The author is grateful to Greg Nelson for sharing his years’ experience in constraint solving, to Greg Badros and Alan Borning for their kind help with the Cassowary solver, and to Henry Sang and anonymous reviewers for valuable comments on the manuscript. The author also thanks the following individuals for their collaborations: Hui Chao, Elsa Durante, Brian Atkins, and Mihaela Enachescu. The personalized postcard system is a joint project with De Young Museum and Digital Pond. Anna Durante and Gary Vondran have championed this research from the start.

References

- [1] Heydon A, Nelson G. The Juno-2 constraint-based drawing editor. Technical Report 131a, Digital Systems Research Center, Palo Alto, California, December 1994.
- [2] Badros G, Tirtowidjojo J, et al. A constraint extension to Scalable Vector Graphics. Proceedings of Tenth International World Wide Web Conference, May 2-5, 2001, Hong Kong.
- [3] Jacobs C, Li W, et al. Adaptive grid-based document layout. ACM Transaction on Graphics, vol 22 no 3, 2003, pp 838-847.
- [4] Purvis L, Harrington S, et al. Creating personalized documents: an optimization approach. ACM Conference on Document Engineering, 2003, pp 68-77.
- [5] Johari R, Marks J, et al. Automatic yellow-pages pagination and layout. Mitsubishi Electric Research Laboratory Technical Report TR-96-29, 1996, <http://www.merl.com>.
- [6] Berkner A, Schwartz EL. SmartNails: display- and image-dependent thumbnails. SPIE Conference on Document Recognition and Retrieval XI, pp. 54-65, San Jose, January 2004.
- [7] Atkins B. Adaptive photo collection page layout. International Conference on Image Processing, Singapore, October 2004.
- [8] Agrawala M, Stolte C. Rendering effective route maps: improving usability through generalization. Proceedings of SIGGRAPH, 2001.
- [9] Badros G, Borning A. "The Cassowary linear arithmetic constraint solving algorithm: interface and implementation," ACM Transactions on Computer Human Interaction, vol 8 no 4, December 2001, pp 267-306
- [10] Chao H, Lin X. Capturing layouts of electronic documents for reuse in variable data printing. Proceedings of the 8th International Conference on Document Analysis and Recognition, pp 940-944, Seoul, South Korea, August 2005.

- [11] Kurlander D, Feiner S. Inferring constraints from multiple snapshots. *ACM Transactions on Graphics*, vol 12 no 4, October 1993, pp 277 – 304.
- [12] Lin X. Active document layout synthesis. *Proceedings of the 8th International Conference on Document Analysis and Recognition*, pp 86-90, Seoul, South Korea, August 2005.
- [13] Lin X, Chao H, Nelson G, Durante E. Active document versioning: from layout understanding to adjustment. Accepted by *SPIE Conference on Document Recognition and Retrieval XIII*, San Jose, USA, January 2006.
- [14] Wang X. Tabular abstraction, editing, and formatting. Ph.D. thesis, University of Waterloo, 1996.
- [15] <http://xml.apache.org/fop>
- [16] Harrington SJ, Naveda JF, Jones RP. Aesthetic measures for automated document layout. *Proceedings of the 2004 ACM Symposium on Document Engineering*, Milwaukee, Wisconsin, USA, 2004.
- [17] Lok S, Feiner S, Ngai G. Evaluation of visual balance for automated layout. *Proceedings of the 9th International Conference on Intelligent User Interface*, Funchal, Madeira, Portugal, 2004.
- [18] Knuth DE. Breaking Paragraphs into Lines. *Software Practice and Experience*, vol 11, 1981, pp 1119-1184.
- [19] <http://www.cs.sandia.gov/opt/survey/mip.html>
- [20] <http://www.w3.org/TR/xsl>