Dynamic Layout Optimization for Newspaper Web Sites using a Controlled Annealed Genetic Algorithm

Gjermund Bø Brabrand

Master's thesis
Master of Science in Media Technology
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2008

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Faculty of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

A newspaper layout mainly consists of rectangles laid out on a surface in a way that produce no gaps, and looks good. The pagination problem for newspaper web sites is trying to make these layouts automatically. We present an annealed genetic algorithm with article control functions to calculate the rectangle position in order to solve the pagination problem. Articles contain metadata that describes individual design requests, and preferred position for each article in categories *headliner*, *float article* and *priority article*. The positioning request tell the algorithm which articles to position first, and which to position around the other to produce a good fit. A fitness function is used to measure good layouts. The method is tested on two groups containing respectively 10 and 6 participants. One group tests the fitness function performance with the control function implemented and the other group tests the prototype which presents a publishing tool with a dynamic layout generator. The prototype test group contains experienced persons working in large Norwegian newspaper web sites. Although we wanted to add control to the automation there were common agreement from the participants that a professional newspaper web site would need a higher level of control. There was also a common agreement that other areas within web site publishing could benefit a layout generator.

Acknowledgements

Thanks to supervisor Rune Hjelsvold for guidance as well as sharing of experience. We would also like to thank all the people participating in the experiments. It is not easy to get people to spend 30 minutes of their time on someone they don't know, and we are therefore grateful for your participation and time!

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Background

What makes the digital environments around us attractive is the illusion of being surrounded by spatial information everywhere we go. Information access has never been easier. Radio, podcasts, mobile phones and of course internet all feed us with information, and we can access the medias from everywhere, either on a laptop, mobile phone or a radio. The fact that all larger cities now expand their wireless network access does not counteract this trend. As the digital era increases its dominance we see an increase in how automatic procedures ingest everyday doings. A natural area of development for all the information available on the internet is e.g. search and organization tools to display it [4].

The computer early became an important tool in desktop publishing (DP). Content such as images and text was easy to design and handle in the digital world. Reuse of templates and digital tools saved many work-hours. DP software made the computer a must have for publishers. This also brought the pagination problem to surface. Pagination is the term for performing page markup automatically. The problem with finding a compact and harmonious positioning of page content is discussed in [5], where automatic procedures are presented to solve the Yellow-Pages pagination and layout (YPPL) problem.

After a significant trend in the end of the 1990's, most newspapers converged with internet in the years after the millennium. This opened for new problems to be solved when analogue experience had to convert to digital environments. Where printed newspapers can change layout and grid for every publications most publishing tools for web (CMS) offer static layout templates. This may be due to lack of web standards at the time of conversion, or simply the constraint we experience in this project. One of the problems with dynamic layout is that it depends on certain control on top of the automation such as supervision or boundaries. In the beginning of World Wide Web there were no such thing as web standards, and this caused problems for engineers developing cross-browser compatible web applications. The use of web standards has increased rapidly the last 10 years.

A web site attracts four different groups of people: the web designer, the web developer, the administrator and the consumer. Web designers provide the last in modern design. Web developers create the application needed to run the requested functionality. When these two have published their work, administrators post content for the consumer. This approach results in dynamic web applications with good design, but not dynamic layout, and therefore not necessarily a layout that best presents the document. This is one of the main challenges of document delivery over the internet [2]. The web designer has the distinct desire to specify the appearance of the document, while the consumer and the browser come with its own limitation.

A web site is often built by multiple rectangles containing text, images, videos, flash and so on. These rectangles are part of a layout created by a web designer to combine good design with good user friendliness. Rectangles are placed in a static grid system where the content in some of the boxes are dynamic. An alternative to placing these boxes in a static grid layout is to place some or all of the boxes where best fitted based on its neighboring rectangle boxes. The problem has been discussed for a long time,

especially in terms of the mentioned pagination problem [6] where automated procedures have been worked out to save thousands of work-hours. The growth of internet as a medium creates new areas for these techniques. The combination of administrator interaction and automated layout procedures can avoid some of the constraints a regular web page can experience.

1.2. Problem description

In this project we suggest a way to equip a newspaper web site with dynamic layout but at the same time keep some of the article control that is needed for a newspaper. We describe article control as the opportunity to select headliner(s) and articles with high priority. This will benefit dynamic web sites, making them appear updated and innovative. Our intension is to create the feel of change for the consumer as well as a content based presentation that adapts to the collection of articles to be presented. By combining different methods in document presentation with an algorithm for layout calculation we present a prototype as a suggestion to solve the problem.

Suggestions on how to solve pagination and automated layout problems have been presented over several years [4][5][8][9][3], and some of them specifically within newspaper presentation layouts [4][3]. Common for them is that they suggest an evolutionary approach for solving the problem. We have inherited these methods, and added functionality to offer object control inside the dynamic layout.

This chapter will explain background theory followed by a description of the Genetic Algorithm and Simulated Annealing, which our solution is based upon. A literature study is presented in chapter 2 discussing relevant research and techniques for our problem, before presenting our method and research questions in chapter 3. In chapter 4 we present the prototype followed by test results in chapter 5. Finally, we end the report mentioning several possible improvements and possibilities for further work followed by conclusions in chapter 7.

1.3. Terms and definitions

Below is a presentation of the terms and definitions used throughout the thesis.

Layout optimization is a term describing the process of finding a good layout efficiently. The visual appearance of a layout is as important as how efficient the content in the layout is being presented. In layout optimization we try to find techniques and methods to create well serving layouts for different occasions.

Dynamic layout means a layout that changes from time to time based on content. Dynamic layout has also been used to describe resizable layouts that adapt content size and area to fill the presentation area. We refer to the change of rectangle positions. No hardcoded grid restricts content updates because the layout will create a new pattern each time the content changes.

Static layout is what most web pages operate with. Content rectangle positions in the layout are hardcoded, but the content inside them changes as the web site is updated.

In this thesis we are talking a lot about **generating a layout**. We try to create layout automatically based on which rectangles (articles) that are to be displayed. The system generates a layout that presents the content rectangles on the least amount of area. This is done on request from administrator.

Front-end and **back-end** are the logically separation terms in web development to determine what the consumer sees (front-end), and what the administrator sees (back-end). The back-end is mostly equivalent with the control panel interface that the administrator uses to publish content on the web site.

Global and **local optimum** is terms used in search optimization. Many search solutions will get stuck in a local domain of the search space resulting in the possibility of not even touching potential solutions. This gives a more random solution, and is called local optima, because the solution is based only on parts of the search space. Global optimum on the other hand is a solution based on the whole search space, and is considered a better solution.

The term **article control** refers to interaction for affecting the positioning and presentation of the articles.

1.4. Motivation

The stabilizing of the Internet as a marketplace after the boom in 2001 has brought a tremendous rise of content management systems (CMS) for web. It has not only become an important channel for sales and marketing, but also an important tool for communication and information flow. As consumers become more and more aware on products, manufacturers and developers also become aware on how to market their products. For a while manufacturers have tried to create products that adapt to the consumer, known as personalized and adaptive web sites. The idea is to give consumers what he or she wants and needs. It is also a matter of making good presentations for consumers. How can we make a dynamic environment that can easily adapt to different situations? Automated layouts try to present content in a way that uses the space effectively. The concept is that content metadata is evaluated in order to create a layout that fit all the content efficiently together, preferably on least amount of output area.

Publishing systems for web today (CMS) help users to easily handle all published material by offering intuitive user interfaces. This makes it easier to update a web site frequently and is very helpful in order to keep time costs down as well as presenting dynamic content for consumers. Still there are certain constraints to the way documents and content are presented. Normally back-end and front-end of web sites are logically separated. Back-end punches articles, photos, metadata etc. to database and do not necessarily interact directly with the front-end. This is because the front-end does not need any direct contact with the back-end. The database is not only the storage compartment of a system, it is also the connection and communication between the back-end and the front-end. There are positive and negative aspects with this. First of all a system will always benefit being split into several separate parts. This eases tasks like updates, design-change etc. On the other hand one might miss out on opportunities that come from a more direct communication between front-end and back-end. Front-end interaction can be integrated into the back-end posting process in

order to adapt the new content to its environment. Examples of this can be seen in e.g. Mootools where front-end "drag'n drop" method on objects opens for administrators or consumers to change the layout on demand. The development process can be challenging for the programmers and web designers to adapt to, and will most often include more work hours. This may also result in web designer and web developer cooperating slightly more than usual.

Automating the layout process of newspapers was discussed shortly after WWW's birth, and was one of the first newspaper processes automated with the help of computers [8]. By wanting to implement dynamic layout which is the top layer in web design, we need to think of the layer as a part of the back-end system. This is because dynamic layout can make a web site vulnerable and needs some kind of supervision. In this thesis we present a suggestion to this problem in terms of a prototype, and run the prototype through some tests. By this we hope to highlight some issues that we believe will be most relevant in future web development.

Draapenett is a small Norwegian company which has been developing CMS and other web tools for customers since 2006. Based on their experience from delivering web sites their hypothesis is that automation may not be used as diligent as it could be. We will use their experience as well as parts of their newspaper web module to present our solution to the problem.

1.5. Automated layout

Automated layout is the procedure where presentation content is processed by a computer or a machine to create its layout. Researchers [9] have described automated layout to be:

> *Automated layout refers to the use of a computer program to automate either all or part of the layout process.*

As information to present increase along with the growth of internet, the amount of data rapidly overtake our ability to present it manually. Automated layout is one of the solutions used to prevent making presentations and layout manually. An area of use can be recognized in pagination, where the idea is to fill a printed page with as much content as possible, avoiding gaps and white spaces. Yellow Pages has worked on this issue for several years, and several firms have tackled the pagination problem using automatic procedures such as the Finish so-called V.I.P^{E} system and the German YPSS++ [4]. In terms of newspaper the problem was already discussed in 1995 [1], where a personalized newspaper presented layout on article-requests from the user. [1] defines the newspaper layout or pagination problem in the following way:

> *The problem of laying out a fixed set of news contained in rectangular boxes, with no overlapping, putting all of them within the limits of the browser, in a minimum surface and in a minimum amount of time.*

In our work we refer to automated layout as the process of optimizing rectangular boxes for websites into newspaper layouts. A computer suggest a layout based on how well all content rectangles fit together. The positioning of each article is decided based on its neighboring articles and article metadata.

2. Literature review

The idea of automated layout for newspaper web sites is not new. The problem with pagination has been discussed by many for many years. This problem occurs when you want to automate the process of laying out information for prints. Yellow Pages is saving a lot of work-hours using automated procedures. This is also what led to experiments described in [4], where the same technique - only modified - is used for effective automated article layout. [4] did some testing back in 1998 using an annealed genetic algorithm (simulated annealing in genetic algorithm principles) for laying out rectangles on a web page. This was done implementing the algorithm client-side using JavaScript. This work was very innovative at the time, and has become even more effective with time since most computers now carry much faster processors. This is highly relevant for our study since part of the problem we are looking into is how to present news articles in a dynamic layout for web site newspapers.

Another research project later presented an improved solution with better results [3]. Here an annealed genetic algorithm is presented for the layout generating. The problem in [10] was that the runtime became too high when dealing with many articles, and the overlapping problem offered great challenges. In [3] this has been solved by adding the constraints of fixed width for articles, so that they are multiple of column width. This made it possible to present layout generating in real time by sending the application along with the web site to the client. This solution is proposed for real time generating of a newspaper based on user requested sets of articles, i.e. intended for a system that receives articles from different channels, presenting them as one new newspaper. A client-side script is suggested in order to benefit individual computer characteristics such as browser and window space available, and to avoid server runtime overload. The basis for the solution described in this thesis is the script presented in [3]. The script is effective for finding an optimal layout for a given set of articles, and is only 15kb. We have not found any research on combining dynamic layout with object control. This will be needed for the control of headliners and priority articles.

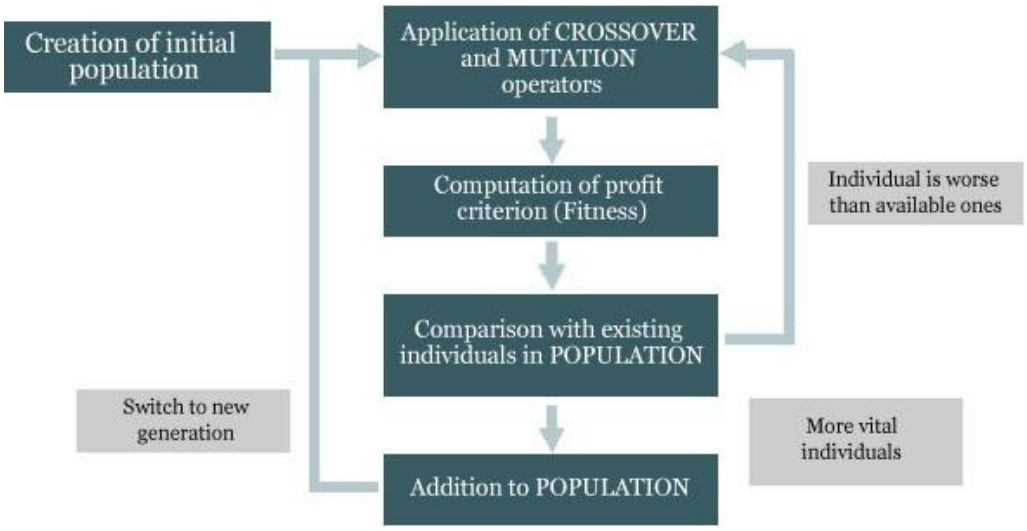
To preserve the quality that most newspaper web sites deliver it is important to control the automation in some way. Reference [9] presents an analysis of techniques used by researcher systems that have automated layout components. Here we find different types of constraints as a way to design automated layout systems. Sets of constraints are used to create networks of object positions and sizes. Abstract constraints describe the relationship between two components and spatial constraints describe positions and size restrictions. A conversion from abstract to spatial is needed in order for a system to read the constraint. Abstract constraints are ideal when dealing with user interaction. In our case this interaction will affect the automated layout, and is therefore in need of a system technique that can translate user requests and parameters into spatial constraints that can affect the system as long as they fulfill system criterions. The constraint technique described in [9] can be used as a way to offer interaction with automated procedures without performance loss.

Constraints have been widely used to set boundaries for system variables. Interaction that will affect the presentation is needed. This means that the administrator needs direct control of system design variables. It is also important that the output could have a fixed design, and that administrators will not affect the parts of the design intended to be static (design profile etc.). [2] discusses the use of constraints in

document layout for web. Constraints are used for multiple operations, among other things designer control of published material. Here content is taken in consideration when creating the output layout. Although we will create a user interface for the administrator to interact directly, [2] will be highly relevant since the same issue occurs when the system handles the content. Constraints are used to set boundaries for the content or documents published on the site. A solution to how the designer can provide multiple sets of the constraints to the consumer is also suggested. Different design templates can contain different design constraints, and will affect administrator actions.

2.1. Genetic algorithm

Genetic algorithms (GA) are used for search and optimization problems to find the best solution in a search space. Genetic algorithms have been used to find suboptimal solutions, and are called approximation algorithms because they attempt to approximate the optimal solution. The concept of GA was first presented by John Holland in 1975 (*Adaptation in Natural and Artificial Systems*). It is an evolutionary approach which is most appropriate for tasks where location of the global optimum is difficult to find. The principle of GA is that it uses the current suggestion as probabilistic to find a better solution. This is due to the fact that in optimization, a bad solution can be very close to a good solution. Where other algorithms are serial and can reach the end point without finding a good solution, the GA explores the solution space in multiple directions which increases the possibility to find a good approximation to the global optimum.



Figur 1: The process of the genetic algorithm. (Originally illustrated by RT SOFT)

The principle of the GA is based on Darwin's "survival of the fittest" [10]. As the name might disclose, the algorithm is inspired by evolutionary biology. The algorithm works stochastically developing generations of solution populations. For each population the algorithm runs a number of generations improving the solution. A solution is called *chromosome* (sometimes called *individual*). A chromosome contains a set of parameters that is a proposed solution to the problem. Based on defining optimization or search criteria the GA uses a fitness function to evaluate solutions, and assign the

solution a score or fitness value. The fitness function is similar to the objective functions used by mathematical optimization process programs. The chromosome parameters are referred to as *genes*. A gene may have different characteristics in different locations inside the chromosome. GA finds the combination of the genes that will result in the best genetic characteristics.

GA can take several different parameters. As mentioned, GA runs a number of generations inside a number of populations to find the optimal solution. It is common to use a parameter to define how many generations to use. Populations on the other hand do not necessarily need a parameter as the GA often is set to run until it reaches a fitness value that is suited for the problem. This only works if a known acceptance value or solution for the problem is known. This is the most important parameter as it will affect the runtime greatly. It is also important when fine-tuning the algorithm to a problem as it can affect the performance from finding a good solution, to not finding any at all. Last but not least is the mutation and crossover probability which defines the acceptance limit of their performance. These two operators are described next.

2.1.1. Mutation

A mutation operator is used to create new chromosomes of the current one, i.e. do small changes to the genes, so that a possible better solution is created. This is an important part of how the genetic algorithm processes itself to an optimal solution. The mutation operators' task is to prevent the chromosomes from becoming too similar to each other, and ensure that evolution is not slowing or even stopped. This prevents what is called the local optima. What it does is to randomly alter one or more genes from its initial state so that an entirely new chromosome is created. Two points are randomly selected (underlined) to switch place:

Original = (1 2 3 4 5 6 7 8)

The mutation makes a new copy of the chromosome, and creates a new one where the two points have been swapped.

Mutated = (1 7 3 4 5 6 2 8)

2.1.2. Crossover

Another GA operator that is based on the idea that a fitter solution is closer to the optimum, is the crossover procedure. Crossover takes two selected chromosomes, and crosses a number of genes at some point, creating two new chromosomes. This procedure fits best to implementations of the algorithm where the chromosome contains binary genes and not reference genes, because crossing references may cause duplicate entries in the offspring chromosomes. There are different ways to do this procedure, but the most common one is the *one point* or *two point* crossover. An example of a two point crossover:

Chromosome 1: (1 0 0 1 | 0 1 1 |0 1 0 0)
Chromosome 2: (0 1 1 0 | 1 0 0 | 1 1 0 1)

Offspring 1: (1 0 0 1 | 1 0 0 |0 1 0 0)
Offspring 2: (0 1 1 0 | 0 1 1 | 1 1 0 1)

As a result offspring 1 has basically become chromosome 1 only with chromosome 2's center genes. Knowing that both the original chromosomes were fitter solutions, the idea is that one of the offsprings now contains all the good genes. Both mutation and crossover use an acceptance probability to define their performance. This mainly defines the allowed difference limit when producing offspring.

2.2. Simulated Annealing

As with the GA Simulated Annealing (SA) is another algorithm used for the global optimization problem. SAs have also been successful in circuit design problems. SAs terms come from metallurgy where metal is heated quickly and then slowly cooled in order for the metal crystals to shrink and reduce their defects. SA uses a *temperature T* to control the process. The idea is that when the temperature is high the algorithm allows disorder and exploration of the search space. As the temperature decreases slowly order is found until T freezes at T=0 (zero). SA uses the energy of the process to decide when to lower the temperature. This point for temperature change is called *thermal equilibrium*. When energy is negative the temperature is lowered, allowing only solutions close to the current one. When energy is positive the solution acceptance increases so that the current solution may be replaced by a worse one. The ground state, T=0, is the same as the global minimum. As with the GA, SA needs to be fine-tuned to the system in which it is intended to be used. This includes setting the initial temperature and the number of populations to perform for each temperature.

2.3. Constraints

Constraints are used as boundaries for what is accepted in a system or a program. The constraints define the criterions that must be fulfilled in order for a solution or task to be valid. Within design and layout optimization constraints have been used to define both requested positioning of object, as well as system criterions that these must follow. [9] presents spatial and abstract constraints as a way to combine system criterions with user interaction. Spatial constraints are described as criterions that hold positioning and size of components, e.g. *caption 1 below text*. Abstract constraints are high-level relationships between layout objects, e.g. *text1 references pic1*. While spatial constraints can be read directly by a system, abstract constraints must be processed by a constraint solver. A constraint solver will translate the abstract constraints into spatial constraints.

Alan Borning [2] propose a technique that lets both author and viewer define document layouts together. He uses required and referential constraints to let the viewer interact with display requests without changing the authors required system constraints.

3. Methodology

3.1. Research Strategy

This thesis uses mainly a qualitative approach, while one of our research questions will be answered based on results from a quantitative test. We use the qualitative *verification* purpose as suggested in [7] to research the validity of theory within real-world context. Our goal is to suggest a change for newspaper web sites, and our hypothesis is that automatic layout would benefit the presentation of articles. By doing experiments on people with related professions we want to determine people's perceptions of a particular situation and try to understand how the suggested theory work in real-life. We found our problem well suited for the qualitative *phenomenological study*. This study has the following characteristics [7]:

Design	Purpose	Focus	Methods of Data Collection
Phenomenological study	To understand an experience from the participants' point of view	A particular phenomenon as it is typically lived and perceived by human beings	• In-depth, unstructured interviews • Purposeful sampling of 5-25 individuals

Tabell 1: Phenomenological study

The results are collected as a combination of a field study, a survey and a literature review. The field study consists of experiments with a prototype and the survey is a web based interview. This experiment is taken by 6 participants with experience from CMS or other similar publishing tools for internet. A criterion is that the participants work with these publishing systems daily. Hopefully we'll be able to discover a potential or possible changes and weaknesses with the prototype based on the individual feedback each of the participants is supposed to respond.

We present a center aligned surface which is 770 pixels wide. Dependent of system settings, this surface can be divided into as many columns as desired, but must be considered along with the column width. For our suggestion we present the system with three columns in order to give the presentation a modern newspaper look. The articles are fixed size rectangle boxes with widths from 1 to *n* times column width (3 in our settings), and is delivered in an HTML file collecting them from database.

In the following section we will list our research questions and how we intend to solve them.

3.1.1. Research questions

This project is based on the following research questions (RQ):

For this question we want to find out how automatic layout can be implemented in a newspaper web site publishing system in order to get the best out of both the automatic layout and the consumer's/administrator's experience.

- RQ1: How can automated layout procedures benefit a newspaper web site advantageously?

We want to find a solution to how headliner and article control can be combined with the annealed genetic algorithm without taking the efficiency away from the algorithm.

- RQ2: How can article control be implemented in the GAs fitness function without loss of effectiveness and performance?

Since we will be using a computer to decide what is a good layout and what is a bad layout, we want to check if the computer correlate with human eye.

- RQ3: How well does the fitness function and human eye correlate in picking out visually approved layouts?

Articles normally fall into static frameworks when published, and administrators does not have to 'be aware' of how it will look front-end. This might change when automatic layout is implemented, and we want to know if adding new techniques to a publishing system will cause any problems in terms of procedural changes.

- RQ4: What positive and negative factors will automated layout in a newspaper web site have on the administrators workflow compared to regular news posting?

We want to know if solutions like the one suggested in this thesis is chosen not to be used or if the market is not aware of the possibilities that lie within automated procedures.

- RQ5: To what extent is the publisher community aware of the potential of content-based layout optimization?

3.1.2. Method

RQ1: As part of developing a prototype we will suggest a system design that is suited for automatic layout in newspaper publishing environments. Starting with a typical CMS design for a publishing module we will adapt this to fit the automatic layout generator and adapt the automation in a way that does not affect the publishing process negatively. A design model based on our result will be presented.

RQ2: Different solutions to how article control can be implemented inside and outside the fitness function of the GA are suggested in our prototype. We will discuss the different solutions and keep the one that best combine algorithm runtime with good results. A good result is reached when article control works advantageous, and the layout presented is approved by human eye, i.e. the algorithms solution is also considered a solution by human. We will measure the algorithms efficiency and performance to find the most optimal solution.

RQ3: By giving a group of participants a set of layouts suggested by the prototype, we will compare the results from the fitness function with the participants' opinion. Participants will be asked to place 8 layouts in order 1 being the best and 8 the worst. This order will be compared to the fitness function. We hope to find correlation between the two sources.

RQ4: Project author experience compared with answers from a survey will form an understanding of how big changes the solution will have on regular publishing workflow. Participants with relevant experience will give their feedback on how they experience the changes.

RQ5: For this question we will do an internet-study using internet as a main source to find out more about how much attention automatic procedures get, and where the discussion is directed. Basically we hope to find a possible trend by visiting web forums and user groups. We also hope to get some feedback from the survey that is relevant for this research question. Because this research question will use internet as a reference, we must be aware of the risk of information not being representative enough as well as the grade of truth. Our goal is merely to understand the interest and awareness of automated procedures for newspaper web sites.

3.2. Problems

Dynamic layout would not fit into all kinds of web modules. List based presentations and HTML form sites will most likely keep their strength staying in a static layout. In these cases a sufficient result could be reached by simply changing the CSS properties to change the layout. We propose the method on a module where the technique is believed to make benefits, namely a newspaper module. A newspaper module is the part of a website that deals with publishing and presenting articles. Implementing a solution that takes all the articles for a web page in consideration and present a layout based on these, could result in more efficient use of window area in addition to a more exciting web site.

The Spanish J. González Penãlver and J. J. Merelo are some of the pioneers within layout optimization for newspaper web sites. They have presented several research projects [4][3] looking at ways to automate the layout process of laying out articles on a web site. Techniques proposed include using genetic algorithm and simulated annealing as client-side script in order to position articles around on the screen using a fitness function to calculate the results obtained. In this thesis the foundation of their work is taken a bit further. In addition to update the JavaScript to be compatible with today's browser technology we suggest two operators in order to obtain the control needed for realistic use on a newspaper web site today. The challenge is to manipulate the automation without losing algorithm efficiency and performance.

We propose an algorithm as a client-side script in this work. There are several reasons for this. From experiencing lots of skepticism from users being afraid of viruses and spyware, enabled scripting setting in web browsers has become common and presupposed by web developers. This can be seen in help technologies and architectures like Mootools [8] developed by Mad4Milk and AJAX, which can be found on thousands of web sites all over the world. These are mostly JavaScript classes that deliver effects and tools to create better web sites. Several other developers offer similar products, and this is a trend that points in a direction telling us that scripting (especially JavaScript) for web browsers is an important part in future web development. On the other hand, a server-side script doing all calculations before site load would ease consumer processing. Compared with a client-side script which processes the layout for every client/consumer, a server-side solution would load the site the usual way; server builds the current presentation and sends it to the client. As long as the layout calculation is an administrator process and not for clients, there is

no reason to expect server overload as was the reason for a client-side solution in [3]. Having this in mind our choice of selecting between a client-side or server-side script to solve our problem fell entirely on user experience and friendliness.

3.3. Pilot Study

The solution presented in [4] was a client-side script that positioned the article rectangles on the screen based on an annealed genetic algorithm. This script was actually very efficient even though it was written back in 1998. Further research [3] improved runtime and efficiency by adding a few constraints. The genetic algorithm presented was annealed, which means that it uses simulated annealing in a genetic algorithm environment. Annealed temperature is used in the genetic algorithm to obtain better solutions, i.e. locates a good approximation to the global optimum. The script uses mutation as operator to prevent a local optimum which would not give a wide and deep search.

3.3.1. Comparison

The solution presented in this project is based on the solution proposed in [3]. This technique is highly relevant for our work. We have inherited the methods used, and done some changes in order for the method to fit into a regular CMS publishing tool for newspaper web sites. The first thing we had to do in order to get the script up and running was to replace some JavaScript (JS) functions that limited the use to Netscape 4.0, which came in June 1997. The script used the document.layers property which is not approved by W3C [12]. They came up with the document.getElementsBy* property and is the standard for manipulation of html elements. Although the updated script is restricted to work only in Internet Explorer 7, the document.getElementsBy* is cross-browser compatible (W3C standard), and it will therefore not take too much effort to make the script cross-browser compatible. These JS properties are the communication path between the layout functions and the layout objects. Further improvements are adding functionality for headliners and article positioning. We will implement two operators; one headliner operator and one swap operator. The headliner operator is a function we are going to implement into the fitness function of the GA, and will include functionality to select headliner article(s) for the presentation. The swap operator is a function that makes it possible to select which articles to display first (high priority articles), and which to blend into the page (float articles).

These are the only modifications that affect the result of the genetic algorithm. Other changes done were to implement the functionality inside a CMS. The original script was delivered with a user interface to execute the layout generator. This has been equipped with a save function that lets the administrator save the preferred layout for the published site. Parts of the script had to be modified into two versions because we only wanted the generating of layout to be reachable for back-end users, i.e. publishers, while front-end only presents the current layout.

3.4. Setting priority articles

A challenge with automatic procedures is to obtain control over the process and be able to affect the result. An important task in any kind of publishing system for a newspaper web site is the option to select which articles have greater priority than others, i.e. positioning the articles. To manage this it means that the automation must be manipulated. We found two alternatives to this problem. A natural choice was to implement the option into the fitness function of the GA since this is where the solution calculation takes place. The Article Builder delivers a list of article ids to show at an early stage ordered from high to low (based on priority). This list will not change based on other calculations done in the fitness function. This means that implementing the option into the fitness function would cause a static operation to repeat itself for every chromosome crated, which again means unnecessary runtime. Therefore we tested two versions, one operator implemented inside the fitness function and one operator doing the operation once on the suggested solution chromosome. This way we could, based on results and runtime, decide which version of the Swap operator that works best. The SWAP operators are explained in detail in chapter 4.5.

3.5. Setting headliner articles

Another important feature is the option of selecting one or more headliner articles. The original script [3] did not include this feature, so we had to find a way to implement a manipulative method to set headliners. Since this feature is very important for the system in order to work realistically, we needed a solution that was sure to turn on or off an article with 100% certainty, i.e. knowing that a requested headliner becomes the headliner. Where *S* is the article width in columns we suggested that a headliner have S = max. This is often used in newspaper web sites since a headliner is supposed to attract attention. With a predefined width on all headliners we know that in order to put this article or articles on the page top we would only have to move the article id to the beginning of the display order. We implemented the headliner option as a part of the chromosome creation process at the same level as the mutation operator, inside the fitness function. The headliner operator is explained further in chapter 4.3.

3.6. Field study

3.6.1. Participants

The prototype test group consisted of 6 participants with relevant experience in the area that we investigate. People who work with or have worked with publishing systems for web have been asked to run some tests on the prototype and then answer some questions. This way we get feedback from potential users with the right experience. By testing the prototype on people with experience from other publishing tools for web we hope to uncover the parts that need further notice, and the parts that may not get the attention we hoped for. The participants in this group represents different professional newspaper web sites having respectively 29 479, 323 455, 1 159

208 and 2 800 000 unique readers a week (numbers from TNS Gallup – week 20/2008).

The fitness function test group was 10 randomly asked persons with average knowledge of newspaper web sites.

3.6.2. Survey

All participants taking part in the prototype test will answer 10 questions explaining their thoughts on the technique we use to obtain automatic layout. These answers are main data source considered in the discussion chapter. The questions aim at finding an overall individual opinion on how this technique could work in a real newspaper web site as well as finding which degree of potential the technique may have in other areas.

3.6.3. Test of prototype

Participants receive guidance [APPENDIX A] by e-mail in form of a PDF file. The guide consisted of 12 simple steps to go through describing in detail how to post an article with the prototype. The participants will recognize the build-up from the experience they have from their respective publishing systems. When testing the prototype they will logically compare their experience with the prototype, and the questions will uncover what works in real life and what does not.

After posting an article the participants are asked to do the layout generating process. After an article post the newspaper waits for a new layout to include it. A layout handling interface reveals itself along with the newspaper front page. Participants can execute the script as many times as he or she wants until a satisfied layout is accomplished. The layout is saved and will now work as the current front page layout. The questions that the participants have to answer are based on this process. The test takes approximately 30 minutes.

4. Prototype

4.1. Technologies

The solution presented in this thesis is developed in JavaScript combined with the WAMP (Windows, Apache, MySql, PHP) technology package. W3 refers to JavaScript as the number one scripting language for web. It is a flexible language with good functionality e.g. when it comes to handling small user requests without reloading the page from the server. This is best known in the AJAX architecture. PHP is a general-purpose programming language for web. We use it to offer the back-end system for administrators to handle the newspaper as well as laying out articles to HTML. The system is built based on regular CMS standards, i.e. a front-end display for the web site content and a back-end for the publishing tasks. The algorithm that does the layout calculation and generating is a JavaScript running in the HTML presentation file. The site content is loaded along with the script from server. The script is executed client-side on site load positioning the articles.

The prototype use Smarty Template Engine to separate code and design. Smarty offers a framework for keeping the code readable as well as giving the system nice separation between design templates and source code. This makes it easier to change templates for different tasks (e.g. design change), and implement the technique to multiple systems.

The back-end module use the rich text editor TinyMCE (WYSIWYG) for editing and posting of articles.

4.2. Usage

In this section we go through the different areas of usage for the technique and the prototype. Our main goal with this project was to implement automation into the presentation process of web site newspaper without losing all control over components. We intended to suggest a solution that could vary the presentation more than what most newspaper web sites do today. An example of this can be explained by looking at how some newspapers (e.g. the Norwegian newspaper web site Dagbladet.no) always uses large headliner photos although the headliner article may not deserve the attention this causes. This can be frustrating because consumers need to scroll down to see the headliner title. Sometimes several articles in the presentation could benefit appositional size so that they get the same attention. One may think that this could be solved by changing the image size, and that every component wraps together based on this, but this is not always the case. In systems where components are moved upwards if there is enough space available, this could be the case, but this is not a normal way of developing newspaper layouts. Most systems use grid-based presentations. This involves constraints in columns per row, and is a static presentation. The prototype offers presentation with dynamic layout. Rows are flexible, while the amount of columns is preset based on the desired presentation area. Articles are placed based on user given metadata that implies how and where an article is displayed in the presentation. This gives the user (i.e. publisher/administrator) the ability to decide whether or not an article deserves to be presented with a total amount of columns width, or if it should be presented appositional with other articles.

The user who publishes articles can choose from three different article types:

- **Float article** *[Default]*
  Articles without any specific placement desire. These articles are important for the system as they are wrapped around articles with requested placement constraints (*priority articles* and *headliner articles)*.

- **Priority articles**
  Articles of a certain importance. These articles are placed under the headliner(s) based on its collective priority order.

- **Headliner articles**
  Articles of great importance. Often the newest published article. Can consist of several articles if required. Presented prior to priority articles.

Headliners can be set by selecting the article type *headliner articles*. These articles are presented prior to all other articles. Based on the width parameter set for each article, the system automatically finds the amount of rows needed to present the articles. Although the amount of headliners is commonly one, the system supports unbounded number of headliners. This gives the user opportunity to select between headliners and *priority articles* (explained below) when building a presentation. This increases the option to place articles relational to each other.

Additional to the ability to vary the way headliners are presented, other articles are also in need of placement control. Next from headliners come the regular articles, still of high importance. These are referred to in the prototype as *priority articles*. Priority articles work much in the same way as headliner articles. They are placed after the headliner(s), and presented based on the individual priority values.

A final article type is the *float articles*. These are very important for the algorithm presented to function desirably. These articles do not contain any specific placement desired from the user, and are therefore used as building blocks around the other article types. As almost no article size matches each other the system will perform better with a large amount of float articles, because the amount of different sized building blocks is available.

The prototype presents the automatic procedures described in this project in two ways; pre-generated front page layout, and real-time generated category layouts for sub-sites. The front page of a newspaper web site must be the same for all users, and is therefore generated in advance by the publisher/administrator. This is part of the article posting process. To present different usage areas for the technique we also implemented real-time layout generation for consumers when a news category such as *culture, sports, music* etc. is selected. This means that the system will present all the active articles for the given category and generate the layout on site load.

4.2.1. Static framework

The prototype only presents the generated area plus a static header. Our intention is to present a system that can be included into a static framework, where only parts of the web site contains dynamic layout. Although our presentations of the prototype do not highlight this usage, it is important to know that the automated procedures presented

are only intended on the article content presentation area of a web site. This is illustrated in section 4.6.1.

4.3. Constraints

The majority of research in automated layout discusses constraint-based methods to solve their problems [2]. There are several approaches to design a system that is intended to let the user control and affect automation. In this work we have suggested a solution where user parameters create constraints as guidelines for the layout we are creating. Constraints have been used for specifying window and page layout for many years. [2] describe relationships among graphics, boxes, drawings etc. to maintain consistency between the data to display and the application. Designers can describe the desired properties for a system that is combined with other system requirements in a predictable way. Within the boundaries given by the system constraints we suggest a solution which also gives the publisher the opportunity to employ constraints to define rectangle layout and position. These are constraints such as image location, column width, article type and indirect affect on header font size. Options like article type affect the display order in the presentation. The user interface of the parameter action is listed below.

Figur 2: The user interface (in Norwegian) of the publishing authors' ability to employ individual article constraints.

Based on parameter combinations only some of the constraints are handled as requirements. System design constraints are defined in CSS variables used by the Article Builder (explained in chapter 4.3.1). It is natural to define these constraints specifically for every design template used. This method is also described in [2]. Predefined settings like number of columns and default generator settings are defined in database.

Let *S* be the article column constraint, and *L* be the article type. Since an article where *L* = 1 is a headliner, then S = max. If *L* > 1 then S <= max. This basically means that by setting an article headline, a constraint on article column span is set. Since *S* is more important than *L* it will have priority above *L*.

4.3.1. Article Builder

Consider *Figure 2* displayed in chapter 4.3. [9] describe the goal of constraint-based automated systems *to take such a constraint network and generate a set of positions and sizes for each of the components in the network*. A system is defined by the types of constraints it uses. Constraints can be described as being either abstract or spatial. Abstract constraints describe relationships between content, while spatial constraints are described as position or size restrictions [9]. Spatial constraints can be handed

directly to the systems constraint solver while abstract constraints must be processed and converted into spatial constraints before used by the system. Basically abstract constraints can be thought of as directions to be considered by the system in some way, whereas spatial constraints are actual variable values that can be read directly by the system. An alternative to constraint-based user interaction would be direct access to layout variables. This increases the user's freedom and ability to decide entirely how to form the presentation. On the other hand there wouldn't be any outcome boundaries, which is not a very safe solution. Users should only get this kind of freedom to a certain point. This is supported with constraint-based interaction as explained above.

In *Figure 2* we display the user interface that administrator meets when posting a new article. These parameters are abstract constraints, and must be processed in order to affect the system. We have developed an article builder class to do this. This class checks the criterion's for the request to become a spatial constraint at the time, and tries to combine as many requests as possible. E.g. an article cannot be 2 columns wide and headliner at the same time. Some combinations are blocked in the user interface. Another example on how abstract constraint is being processed is the image check. In modern publishing system a trend in using horizontally cropped images is becoming more and more obvious. Article Builder checks the shape on images in order to find the best suited wrapping method for the text. An article width is a multiple of the column width defined in the system settings. Height is expandable and will adjust based on text amount and image size. The script finds article heights using JavaScript element.offsetHeight property on HTML elements recognized as articles.

Abstract const 1	Abstract const 2	Abstract const 3	Spatial const		
			Image	Header	Common
Headliner					
	None	Horizontal img	Width: max	+ 3	Article position: first
			Display: block		
			Align: top		
	None	Standing img	Width: 1 column/2		Image align: left/right /top/bottom
	None	Normal img	Width: 1 column		
Float article / Priority article					
	Img position: top	Horizontal img	Width: 1 – max columns	+ 1 or 2	Align: top
			Display: block		
			Align: top		
		Standing img	Width: 1 column / 2		
		Normal img	Width: 1 column	+1	
		Normal img & Colspan: 1	Width: 1 column	+1	
			Display: block		
			Align: top		
		Normal img	Width: 1 column		
	Img position: bottom	Horizontal img	Width: 1- max columns	+ 1 or 2	Align: bottom
			Display: block		
		Normal img & Colspan: 1	Width: 1 column	+1	
			Display: block		
		Normal img	Preferred width: 1 column	+1	
			Display: block		
			Alternative width: max image width		
		Standing img	Preferred width: 1 column / 2		
			Alternative width: max image width		
	None	None	Width: 1 column / 1.6	+1	
			Align: auto		
Priority article					Article position: under headliner

Tabell 2: Article Builders rule map

Comments on *Figure 2*:

The table displays the rule map that the Article Builder uses to solve abstract constraints into spatial constraints. The blue columns are abstract constraints to consider while the white columns are spatial constraints defined as a mixture of system settings and article metadata.

As a summary we can say that the Article Builder equip an article with both suited and requested visual appearance. The builder will try to fulfill as many requests given by the parameters as possible. The result is that the system knows what CSS values to add to HTML article rectangles. Inline CSS is used for this operation since CSS variables are not yet supported.

4.4. Technical Solution

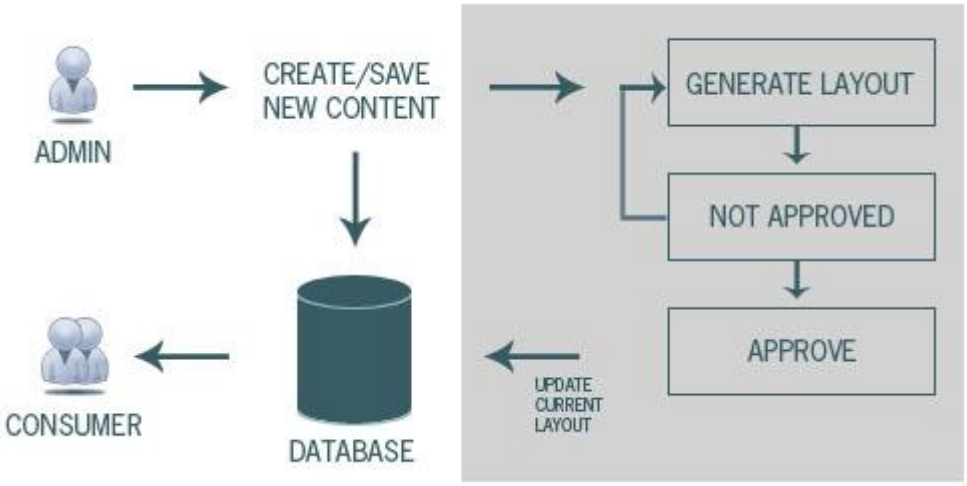
In order to test the method suggested in the project, a prototype is developed to present the dynamic layout generator. The work presented in [3] is our starting point for development. Important functions for article control are developed in order to create a solution that is considered as a modern publishing system for web site newspapers.

We started this phase looking into system designs and requirements for our prototype. The prototype is developed based on regular CMS architecture, i.e. access control layer, database abstraction layer, document management layer and presentation layer. First of all this is because we needed a framework that could present the automation and interaction that we wanted to implement in a good way, and secondly because publishing systems for web are mostly based on CMS architecture. Requirements, such as easy presentation interaction and control, affected our further development. The presentation layer is equipped with a presentation script which presents the dynamic layout. Back-end users, i.e. system administrators, can communicate with the presentation layer in order to preserve some of the administrator presentation control known from professional publishing tools, i.e. Escenic etc. This is solved using the Article Builder explained in chapter 4.3.1. A publisher's article parameters are taken as abstract constraints to a constraint solver which converts them to spatial constraints considered in the presentation. This way administrator can adapt the presentation layer based on requests. The user interface (UI) for admin interaction and article control was presented in *Figure 2*.

The presentation layer, which is a part of both the back-end and front-end of the system, consists basically of an HTML file containing the current article selection to present, and a script containing the genetic algorithm together with presentation- and help functions. Back-end users use the shared functionality to create a layout to store for consumer presentation. The automation is used to generate dynamic layouts for the user, but controlled and only run by administrator. This way we get a system using automated procedures to create a dynamic layout environment at the same time as consumers sitting on different computers will meet the same web site, preventing the feeling of an accidental layout. One of the reasons that we choose to omit consumers to direct exposure to the automation is the safety issue, both in terms of visual quality and technology compatibility. Because automated procedures may cause unwanted results and we present the technique on the presentation layer of the web site, it is important to implement human supervision to the process. Without supervision of the generated layouts, the web site provider would lose too much control since the solution is not 100% safe, i.e. it will not give 50 approved layouts on 50 runs. This is why we suggest an administrator click-through function. This means that the administrator post an article and then uses a front page generator to find a layout to store and use as the current presentation. The system then updates the collection of articles to be presented on the front page, including the recently posted article, and displays one window with the front page just generated, and another window with a UI for running the algorithm again until the administrator is satisfied with the layout. The chromosome of the approved solution is then stored along with an object width list to present the current web site layout.

The current presentation will use whatever layout the administrator last stored on server. There are multiple ways to create this presentation. We suggest a presentation

using the same functionality as the back-end layout generator, i.e. a JavaScript laying out articles in the order given as parameter. The solution layout could be computed once and stored on server instead of just the article order. This would e.g. prevent issues with JavaScript compatibility. This is suggested as part of this projects further work section.



Figur 3: Prototype model design. The grey box is the implementation of the automatic layout functionality.

4.5. Algorithm

For the layout optimization problem we use an annealed genetic algorithm (AGA). The combination of the two algorithms (simulated annealing and genetic algorithm) has been successful for similar problems before [3]. We have altered the output area of the algorithm, and added two control operators in order to provide the algorithm with wanted article control.

This algorithm works on the upper left corner of each article rectangle and measures the width and height. It calculates the positioning of the articles to display which gives the least total gap between the articles on a smallest amount of area. This means that the articles are moved to positions that give best fit among a given set of articles. One can think of this process almost as the classic Tetris game where boxes must be placed together trying to prevent gaps. A good fit is when the articles do not contain unnecessary gaps between each other. AGA uses a fitness function to determine a good result. The fitness function is equivalent to total area used. The chromosome genes consist of article references. The chromosome proposes a solution by ordering the genes the way the articles eventually is supposed to be presented. Each gene's metadata is collected from the server on site load. A list of objects column width is also stored. This list is along with the chromosome necessary in order to implement the solution to a presentation. By knowing the references id order and the correlating widths, the script have the formula it needs.

Figur 4: Example of a possible mutation outcome.

4.5.1. Annealing

As mentioned in chapter 2.2. simulated annealing uses a temperature variable to define the size of accepted solution variance. The temperature functions are calculated as in the reference script [3] (originally defined by S. Kirkpatrick). The initial temperature is calculated:

$$T0 = -\frac{\Delta C}{1 - n(p)}$$

ΔC is the average increase of measured area used, and p is the acceptance probability defined in the system settings. The function that lowers the temperature and thereby narrowing the search space is defined:

$$fT(T0, v) = \frac{T0}{1 + v}$$

where v is the current generation. The algorithm runs until the temperature reaches its minimum. The parameter defining number of generations (g) to run the algorithm is used to define the temperature minimum, and thereby also the final number of iterations. Minimum temperature is defined:

$$Tmin = \int T(T0, g)$$

4.5.2. Mutation and crossover

We implemented a mutation operator to the process of creating new chromosome for the fitness function. We found that the combination of mutation operator and the annealing acceptance of worse chromosomes created reliable results. The pilot project [3] mentioned the crossover function as a part of their implementation, but had not implemented it in their final prototype. Because most GAs uses both mutation and crossover operators and because research where SA uses crossover operators has been accomplished [13], we developed a crossover function to fit the project. We found several challenges with this, causing us not to use a crossover operator in our prototype. First of all an implementation of crossover would be hard in this project because we place article references in the chromosome genes instead of binary values which is probably the most common use. With binaries only two alternatives are possible (0 and 1), but with article references we meet the problem of n amount of alternatives since the chromosome only carries distinct reference values. This could cause duplicate values in the offspring chromosome created by the crossover function, and is therefore not an option. Another possible reason for the crossover operator to be excluded from the pilot study is that it may interfere with the implemented annealing strengths, i.e. crossover is replaced to benefit simulated annealing strengths. This justifies the combination of SA and GA where the annealing acceptance operator and the genetic mutation operator solve the problem presented together.

4.5.3. Process

The layout optimization process is defined in the following code as presented in [3] with additional headliner and article positioning control.

```
initTemp = -(10/Math.log(acceptance probability))
minTemp = initTemp / (numPopulations + 1)
n = 0
//    Create the initial solution
current.solution = initChromosome(numArticles)
current.solution.calculateFitness

   repeat
      for j = 0 to numGenerations
        optimize = current.solution.clone()
        optimize.mutate()
        //  update suggested chrom with priority articles
        current.setPredef(current.solution)
        //  update suggested chrom with priority headliner(s)
        current.setHeadliner(current.solution)
        opt.calculateFitness();
        Δ = opt.fitness - current.solution.fitness
        if (Δ < 0 or random( 0,1 ) < Math.exp( -(Δ / t) )) {

          current.solution.kill()
          current.solution = optimize.clone()
          current.solution.fitness = optimize.fitness
       }
        optimize.kill()
        optimize = null;
      }
      t = initTemp / ( n + 1 )
      n++
   until (t > = minTemp );
```

An example of a typical process for how the genetic algorithm is used in our solution is described in detail:

A collection of articles is laid out in an HTML-file, and the articles reference ids are sent along with their metadata such as size and preferred positions (if any) to the algorithm script. The algorithm starts with the initial chromosome, say [1, 2, 3, 4, 5, 6, 7, 8]. For each chromosome the algorithm use the fitness function to measure height and width and find the list order that will give the best and most efficient layout for this set of articles. It is the amount of text in each article box along with the image that defines how high an article rectangle is. The width is preset through article metadata by the Article Builder. Along with the chromosome the algorithm hold a reference list; say [0, 0, 1, 0, 2, 1, 0, 0] to keep track of each articles column width. 0 equals 1 column, 1 equals 2 columns and so on.

Regular GAs only replaces a current chromosome with a better one, i.e. a fitter one based on the fitness function evaluation. In AGA a chromosome can be replaced even by a worse solution. The annealing temperatures acceptance probability widens the search space by sometimes approving worse solutions. This way the algorithm avoids getting stuck at local optima. The annealing *energy* is defined by the current iteration.

The script takes two parameters; the number of populations to run, and the number of generations in each population. The population is used set the initial temperature and it is therefore more correct referring to *how many generations for each temperature*, rather than *generations per population*. As the code example above displays, the two control operators suggested in this project is executed for each new generation. We initially tried running *Suggestion 1* outside the iteration process, in order to save runtime. Because we work based on the principle of changing the order of the articles to present, we had to find correlating articles to switch place in order for us not to destroy the layout. This was not very efficient because the probability of finding substitutes was not high enough. *Suggestion 2* on the other hand moves the preferred articles to the beginning of the chromosome in the beginning of each generation process. This makes the algorithm calculate the rest of the layout based on our requested starting point. This is a much better solution because we are ensured actions.

When the annealing temperature freezes and an approximation to a global optimum is found, the solution chromosome together with the article column width list are stored as the current layout to display.

4.5.4. Fitness Function Process

As mentioned earlier the fitness function separate good solutions from the bad solutions. It is the heart of the GA and measures how close each chromosome comes to solving the problem. For each chromosome generated the fitness function gives a judgement on the solutions quality. This is calculated based on the solution's total height combined with penalty for gaps between articles. Since the GA proposed in this thesis is *annealed,* it will replace the current solution not only with better solutions, but also nearby solutions. This way the solutions will change almost randomly on a high temperature (t), and will increase as t reaches zero. This is, along with the mutation operator, a contribution to avoid local optima. Because a bad solution often is very close to a good solution, not rejecting all solutions worse than the current one, will widen the search space, increasing the possibility to find the global optima.

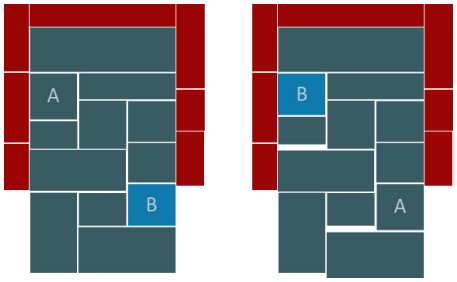
4.6. Swap operators

There is not much literature about manipulation and interaction of automated layout procedures. Solutions are most often used for problems related to the bin-packaging problem, where objects must be packed into a limited number of bins. Instead of bins we have multiple columns to fill. We propose a way to add functionality for article control. This absence confine the realistic use of the technique because this level of control is central when dealing with web site content such as articles. We've tried out two different solutions, one implemented into the fitness function, and one post fitness function operator. Both solutions have their weaknesses and strengths. We'll discuss both of them.

4.6.1. Suggestion 1

The post fitness function SWAP operator searches the chromosome for other articles with approximately the same size, and swaps these two if the target article has a better (earlier) index in the presentation order. A limit l which is defined in the system

settings is used as maximum aberration between the two articles that swap positions. Given a chromosome of [2, 6, 4, 3, 8, 5, 1, 7], let's say that 8 is a high priority article. The swap operator will search all the articles with better positions than 8, i.e. 2, 6, 4, and 3, and look for a match. Given that *limiter* is the maximum aberration, a match is an article with the same column width as 8, and is *l* pixels higher or lower than 8. A large *limiter* value increases the possibility to find a match, but at the same time the visual error becomes greater since a higher *limiter* value will generate larger gaps between the swapping articles. The *limiter* system setting should be defined based on the current design template. This is because some designs will allow larger gaps before they become relevantly noticeable.



Figur 5: B is a high priority article which is looking for a better position (higher up). A is approximately the same size, so they swap places. The area marked red is the web site static framework (commercial banners, menus, links etc), while the blue area is the one with generated dynamic layout.

Pseudo code example:

```
limit = 50      // difference constraint is set to 50px

for ( all articles in priority list ) {
request height = height of request article

      for ( all chromosomes ) {
            target height = height of target article
            p = index of( request article ) in chromosome array

            if ( request article width = target article width and
              target index > request index ) {

                  if ( request height <= ( target height + limit) and
                    request height >= (target height - limit) ) {
                        swap request article and target article
                        in chromosome array
                  }
            }
      }
}
```

In order for this suggestion to find a solution the following constraints must be fulfilled:

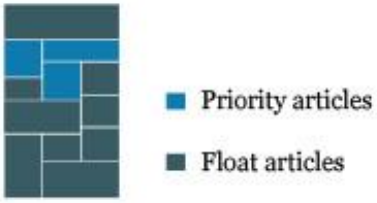
- C1: Request article width == Target article width
- C2: Request article index > Target article index
- C3: Request article height <= Target article height + 50 & Request article height >= Target article height - 50

This is not bullet proof since a) it may not be other articles in the collection with a suited size (C1 & C3), and b) the target index may be better than the original index (C2), but still not good. If all these constraints are fulfilled, then a swap is possible. Following is a list of factors that affect the operator either positive or negative:

+ System design settings such as CSS contains boundaries on how much the rectangles will differ in size, and thereby contribute to create rectangles with approximately the same measurements.
+ The system has a constraint on how much text that can be shown in the presentation (ingress), and thereby prevent uncontrollable article presentation sizes which could increase the amount of distinct rectangles.
- When no swap is possible, a high priority article could end outside the desired area (i.e. consumer must scroll to see article).
- The operator is called after the fitness calculation is finished which means that the layout fitness is only almost right since a few changes is made on the originally calculated layout.
- When many articles are rated as priority articles this may lead to an area battle resulting in bad results.

4.6.2. Suggestion 2

The second suggestion to the article control problem, and the one we ended up using, is implemented in the same way as the mutation operator: run for each chromosome. A list of all the priority articles is created based on article metadata. Articles represented in the list are moved to the beginning (but after the Headliner if any) of each chromosome. The rest of the articles are positioned based on this starting point into the finished layout. This solution takes less processing but in retribution it does not only run once compared with Suggestion 1. The following factors should be considered with this solution:

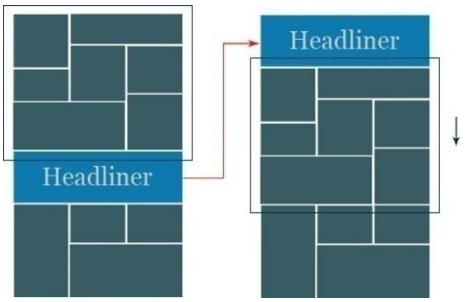


Figur 6: SWAP operator 2

+ The swap operator is considered in the layout fitness score. It is compared with the first suggestion where the layout fitness is an approximation.
+ The suggestion takes less code.
- The operator will perform the swap regardless of the fitness functions' ability to find a good layout. This problem will only occur when majorities of the article collection have a priority attribute.
- In order for this solution to work efficiently priority articles should not be more than 1 column wide.

4.7. Headliners

A good thing with a headliner is that is always comes first. Technically this means that we could just manipulate the presentation order string, moving article id's so that if article id = 4 is a headliner presented in a group like [1,2,3,4,5,6,7] we could just change it to [4,1,2,3,5,6,7]. A position change like this would mess up the layout generating large gaps because the rectangles will lose important building blocks. To solve this issue we decided to put the constraint on headliners saying $S = max$ columns wide. This way the rectangle would not displace the other rectangles when moved to the top. This solution also supports the fact that a headliner is supposed to attract attention. A high priority article can replace a one column headliner if $S = max$ is not suited for the given article, i.e. the article does not "deserve" to fill all columns, and should be appositional with other articles.

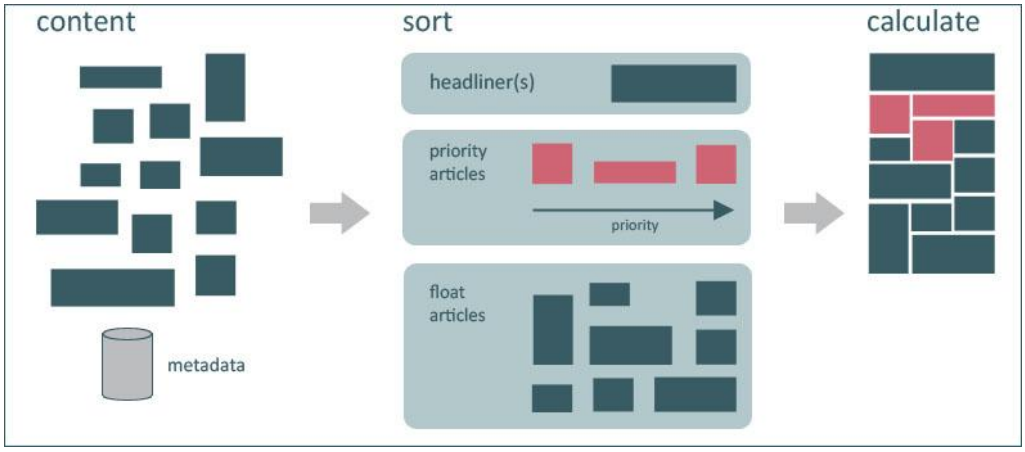


Figur 7: Illustration of how a headliner rectangle is moved to the top after the algorithm layout generating. The gap from the headliner is filled by moving the remaining articles down.

5. Results

5.1. Contribution

In this thesis we have presented a method to affect automated procedures for dynamic layout by implementing two article control operators. These are run by user interaction translated by an Article Builder to affect the final layout. With this work we make it possible for newspaper web sites to create dynamic and exciting layouts for consumers. The combination of techniques and solutions suggested in this project has never been done before, and is hopefully a building block for further research in order to develop timesaving and advantageous dynamic layouts for web sites.



Figur 8: The principle of the project solution.

The principle (Figure 8) of our prototype can be explained by putting the generating process into three groups; *content, sort* and *calculate*. The content group consists of all active articles that the publisher has posted for presentation, along with metadata describing each articles function, requested position, and layout. This is sent to the generator which will sort the collection of articles into separate groups with separate functions. This creates the order needed for the calculation to make a presentation based on user request and appearance quality (fitness).

5.2. Survey

Already early in the process of developing a prototype we knew that control and interaction would be important factors in order to suggest a newspaper web site with dynamic layout advantageous. This is also confirmed in our experiment results. Users experience that article control (definition of headliners and priority articles) is not enough to possess the control needed to run a newspaper presentation. Although the majority of the participants agree that the generated layout is visually good enough, they miss the opportunity to control the articles under the layout generating process itself. Either this may be because they don't feel their interaction in advance gave the result they wanted, or this can be a result of them being intimidated by the automation. Web journalists and administrators are normally used to have control of every aspect of the news posting. Participant 1 compared the solution with Escenic which is a huge Content Management Software offering an advanced back-end

software for professional publishers. From this application, users are familiar with articles' metadata and criterions, but not the layout generating process.

When participants were asked if they missed anything with the prototype there were several suggestions. Participant 1 missed the opportunity to control the *float articles*, i.e. the articles that do not have any positioning options such as headliners and priority articles. Participant 2 on the other side would like a solution building on traditional layout where automated layout procedures fill the gap around manually positioned articles. This is theoretically what the prototype does; presents priority articles under a headliner, and then fill up the rest of the presentation with float articles. Participant 2 on the other hand would like the controlled area to affect not only the presentation top, but separate areas with gaps that can be filled with floating articles. Participant 4 stated a similar suggestion, where the technique could benefit being implemented into sections. A, B and C sections could then offer dynamic layout in each section based on priority type A, B and C.

All participants agreed on question 4 (Appendix B). The generated layout was pleasing and good. Participant 2 stated that he was surprised of how well the system presented the articles. More gaps and random positioning were expected. Participant 3 preferred having the system create the layout instead of him as it would be too time consuming to do by hand. Participant 6 stated the importance of being able to connect related articles together. This is possible to some extent, but then by putting similar priority values on two priority articles. The articles would then be positioned as neighbors.

Participant 4 suggested that dynamic content should be an option in a larger publishing system. His experience from several different publishing tools showed that there are always some unnecessary and irritating constraints in a system, and suggested a combinational solution that could meet with more users.

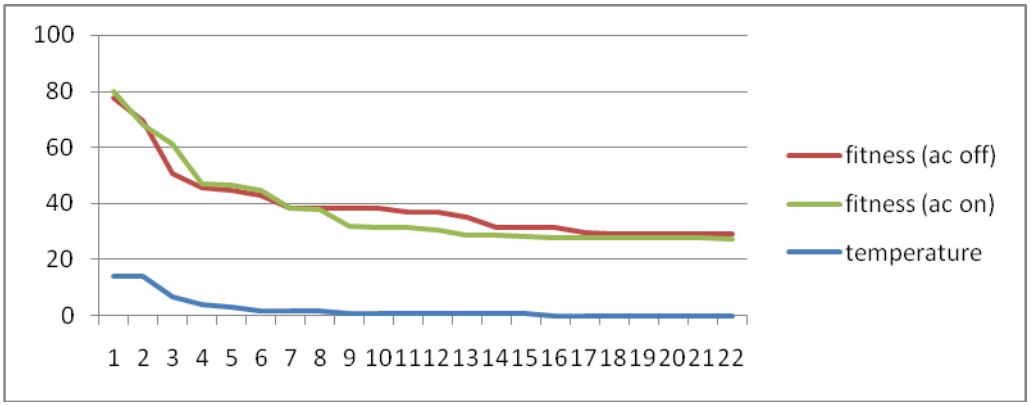
5.3. Technical

A demonstration of the prototype is running and available from prototype.draape.no (login: *guest/layout*). From here, you can download the project source code. The system presents a stripped down publishing tool that presents articles based on content and metadata in a dynamic layout. The article control *(Figure 2)* gives administrator enough options to be able to create a personal feel to the presentation. Using horizontal images along with the alignment: top option, one can decide the look and direction of the layout. The article type option makes it possible to move articles based on priority. Publishers can choose how they want to use the presentation interaction in order to affect the presentation.

Figur 9: Aftenposten.no 9th and 10th of April 2008 shows how the content has changed while the grid is the same. To the right the content is the same, but with two optional layouts.

In *Figure 9* a comparison of the Norwegian newspaper Aftenposten is done. Notice how the layout changes in the prototype, while Aftenposten' layout is static. This presentation of the prototype uses a mixture of all the article control options. It also uses a contrast background color to highlight gaps and positions. A natural choice (when not in demonstration mode) is to use a background color close to white in order to bind the articles closer together. Alternatively a soft border around articles could be used to compromise separation of articles.

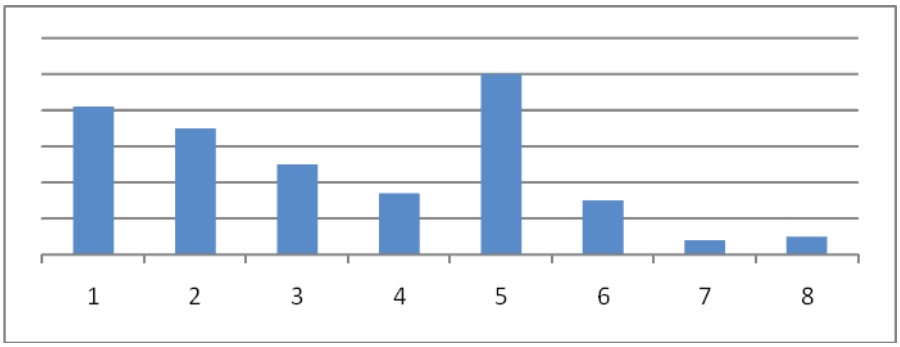
We expected slight changes in performance in our solution compared with [3] due to the fact that we implemented two new operators giving the algorithm less latitude. *Figure 11* displays 2 x 10 runs with acceptance probability $p = 0.8$ respectively one run with article control turned on, and one run without. As you can see there is no noticeable difference in performance. Runtime is slightly longer, but both averages suggest the same solution. In other words the article control operators do not seem to affect neither performance nor runtime noticeably. Runtimes increase with 0.05 seconds in average when the control operators are used with system settings for populations = 20 and generations = 5. The runtime average is 0.35 seconds.



Figur 10: 2 x 10 runs where article control operators are both used and not used. X and Y axis is respectively population and fitness.

5.3.1. Fitness Function

As explained earlier in the report we added headliner and article control functions to the fitness function. The SWAP operator decreases the search space since more constraints are added when certain objects position become predefined. We tested the results given by the algorithm with 10 test persons to find out if the fitness function produced human eye approved layouts. 8 layouts where presented, and the participants were asked to place the layouts in order from 1 to 8. We then compared the order given by the participants with the order given by fitness function. The result is given in *Figure 12*.



Figur 11: Comparison between fitness function and human eye.

The table lists the fitness function order from left (1) to right (8). The blue soils show the participants average on where they placed the different layouts. The table clearly shows that the fitness function score reflects human opinions. Still this selection contains one exception. Layout number 5 was picked as the best layout in 7 / 10 cases. Except for the layout containing one small gap in the left column, the reason that this layout did not get a better fitness score is not known.

We suggested two ways of solving the headliner and article control problem. Theoretically this could be done as a post process (described in section 4.5.2.) of the fitness function since the list rendering that takes place is the same on every run. The downside of this is that the fitness function score would not be 100% right without including these operators. Instead of the calculation and operator talking together, they would do their work separately resulting in misunderstandings and bad results. Therefore we implemented them into the fitness function (described in section 4.5.1.) on the same level as the mutation operator. This way the fitness function would give us feedback on how the new operators would affect the runtime and result. We found that the fitness function use an average of 0,050 seconds more when the operators are implemented into the fitness function process contra when they are run after the fitness function.

5.4. Overall

5.4.1. Layout change rate

A publisher will often consist of several departments cooperating to keep a newspaper web site constantly updated in several categories and media. This will cause much back-end traffic which again will involve the layout generator process. As for larger publishers (NyTimes.com, Iht.com, Aftenposten.no) the layout change frequency would become too high in too little time. As discussed earlier dynamic layout has the power to create presentations that appear updated and flexible. On the other hand, when the layout change rate becomes too high this may cross a limit and become confusing for consumers.

5.4.2. On other modules

The prototype suggested offer an automatic dynamic layout generator where the publisher (administrator) generates the layout to use. An article can be defined as headliner, priority article, or float article. Although the technical solution is adapted to a newspaper module, the technique could possibly benefit other modules as well. Rectangle-based presentations where individual positioning is not necessary such as gallery album presentations and webshops may benefit the solution. In webshops the article control can be used to highlight campaigns and attract attention to certain products. On the other hand this technique should not be used where symmetric appearance is more important than variation, because rectangle sizes will vary too much.

5.4.3. Accessible knowledge

An in dept internet search after literature, research articles and discussion forums revealed that this subject has not gotten the attention it deserves. As for dynamic layout in terms of newspaper web sites we could not find any research done outwards the resources mentioned in this project. Resizable layouts are widely discussed as a method of dynamic layout, but in these methods the automation is directed at fitting content to multiple screen resolutions, but with the same static grid layout. It is no surprise that this kind of dynamic layout is more discussed because it is a lot easier to implement than dynamic layout as described in this thesis. The lack of literature around dynamic layout on web sites may come from several reasons. Techniques have been inherited from computer tasks doing paper pagination. The pagination task can be used on web sites too. This is yet to happen, and hopefully there will be more research done in this area.

5.4.4. Personalized web sites

The research in [3] suggested dynamic layout for a personalized web site. The same method as proposed in this project is used, and calculates a web site in real-time based on articles coming from several different sources. That research does not need to add any given positioning to articles since the requested articles is collected on user request, and is equally prioritized. The work presented in this project could benefit similar situations, giving the presentation automation the possibility to pick some articles to presents as top priority based on interest defined in a user profile. The suggestion could easily be adapted to run based on user profile terms. Terms are used

to define what web site content describes. A generator crawl visited web sites collecting frequently repeated words. This way the computer can learn to know your interest without you having to lift a finger. A technique like this could be implemented in this project's prototype in order to present a personalized web site newspaper, having it look like a newspaper, and even show the articles that appears to be most interesting as real headliners. Theoretically the automated layout procedure presented in this thesis would fit into adaptive web sites quite good, because the dynamic layout could present a lot of different templates, offering a lot of different layouts and at the same time offer computational positioning control in real-time.

6. Future work

Further development with the prototype will include banner support. One of the most important income sources for web sites publishers is the commercial banners presented among other web site components. These should be included among the articles so that they blend with the layout. This should not take too much effort since a banner could e.g. be a two column horizontal article, and would then be handled by the genetic algorithm in the same way as the rest of the articles. The version presented in the prototype is restricted for usage inside a commercial framework, or a framework consistent of both commercial material and static application modules.

We have suggested a solution that runs a script client-side for both administrator and consumer. As part of the further work on this issue we suggest that the consumer presentation is created server-side, and loaded finished into browser. This is because the layout to use is already stored on the server, and by this we avoid the use of client-side execution of the JavaScript in order to present the newspaper front page.

The prototype contains a bug while waiting for a newly posted article to be included in the current layout. A block must be developed so that posting new articles does not duplicate the current article order ids.

A system as the one suggested is normally supposed to be developed to fit all thinkable user requests within certain areas. One needs to look for user habits among multiple professions in order to find what the user needs. We suggest that further work includes the possibility to store old layouts, and archive older versions of the front-page so that the administrator can go back in time. This would be great for several purposes, e.g. it would create a very innovative service for consumers.

7. Conclusions

7.1. Summary

A web designer sees things from a very different angle than the web developer. The same applies for web site journalists using publishing tools for web contra engineers developing the system. A majority of the prototype test participants appeared calm and satisfied posting the article, and a bit more stressed when they met the layout generator. Although efficient and exciting layout is a good thing, the administrator workflow tends to depend a lot on controlling every part of the publishing process. The dynamic layout brings in new issues to consider, and these areas may not belong in the web administrator's workflow, but in the designer's workflow.

As for the algorithms performance and runtime the results were pleasing. The article control operators suggested in this project did not have any negative effect on the algorithm's performance. A slight increase of runtime (0.050 second) is not a problem knowing total runtime on 15 articles was from 0.1 - 0.4 second. The operators are not dependent on calculation, and works well for simple controlling of positioning. Based on this we believe that the technique itself offers pleasing results.

When it comes to answering RQ1, results point in two directions, but with a logical split. It seems as the proposed solution does its job well. Participants were positive to the automation and the result, but could not see it as a replacement of their respective systems. However, this was not the intention, as the prototype is a stripped down suggestion to dynamic layout in newspaper. Anyway the majority of the participants agreed that it could benefit smaller newspapers. This could be understandable when thinking of how much money a publisher can earn presenting its content the right way. Automation could restrict possibilities for niche adaptation. Because of competition and need of quick changes the need for presentation control increases even more. Smaller publishers don't have this rush, and may benefit their web site with dynamic layout as it makes the presentation vary in shape and grid. The solution would benefit sites that are not dependent of moving elements to specific locations. And this puts us in another category of publishing, i.e. newspaper publishing is not in need of automatic layout as the common need of individual control from day to day is too large to be solved by automation.

7.2. Capability

In this project we define our target group as newspaper web sites. This is a big category, and the suggestion would not benefit all newspaper publishers. A professional newspaper is not "all about" automation, but control and positioning of articles. Automated procedures have mainly been used for time consuming problems. Although generating a layout manually is time consuming, the need for control in professional web sites is too big for dynamic layout. The suggested generator could most likely benefit a professional newspaper web site in sections of a presentation, but not as carrier of the main presentation. Based on this we conclude that although improvements and ideas for further work are not absent, professional newspaper web sites will not benefit usage of an automatic dynamic layout generator. Smaller publishers on the other hand would most likely benefit dynamic layout.

We suggest that the technique presented is tried in context of personalized newspaper web sites building on RSS. As we have worked with this project and its research questions, we have seen that automation cannot benefit regular professional newspaper webs sites simply because it could be done manually, and this is a better way to work out the need for control. The static grid layouts for newspaper is therefore a solution that will most likely stay in usage, and the problems caused by them may not be too large compared with problems caused by alternative solutions. The technique could combine consumer profiling with RSS to present personalized newspaper web sites. The profile could then use the control operators to highlight articles containing terms matching the consumer profile. We believe automated procedures as the one presented in this thesis is better suited for this usage because manually control is not an option, and the content will vary based on users. We suggest that further attention to dynamic layout is directed to personalized newspaper web sites.

8. References

[1] Bharat, K., Kamba, T., and Albers, M. 1998. Personalized, interactive news on the Web. *Multimedia Syst*. 6, 5 (Sep. 1998), 349-358.

[2] Borning, A., Lin, R. K., and Marriott, K. 2000. Constraint-based document layout for the Web. Multimedia Syst. 8, 3 (Oct. 2000), 177-189.

[3] González, J., Rojas, I., Pomares, H., Salmerón, M., Prieto, A., and Merelo, J. J. 2001. Optimization of web newspaper layout in real time. Comput. Netw. 36, 2-3 (Jul. 2001), 311-321.

[4] González, J., & Merelo, J. J. 1998. Optimizing web page layout using an annealed genetic algorithm as client-side script. In *Parallel Problem Solving from Nature — PPSN V*, 1018-1027. Springer Berlin / Heidelberg.

[5] Johari, R., Marks, J., Partovi, A. & Shieber, S. Automatic Yellow-Pages pagination and layout. In *Journal of Heuristic*, Vol 2, Num 4, 321-342. March, 1997.

[6] Lagus, K., Karanta, I., and Ylä-Jääski, J. 1996. Paginating the Generalized Newspaper - A Comparison of Simulated Annealing and a Heuristic Method. In Proceedings of the 4th international Conference on Parallel Problem Solving From Nature (September 22 - 26, 1996). H. Voigt, W. Ebeling, I. Rechenberger, and H. Schwefel, Eds. Lecture Notes In Computer Science, vol. 1141. Springer-Verlag, London, 594-603.

[7] Leedy, Paul D., Ormrod, J. E. Practical Research – Planning and design, 8th Edition, 133-160.

[8] Lie, H., Bender, W. 1992. The Electronic Broadsheet: all the News that fit the display. In *Proceedings of Electronic Publishing*, Cambridge University Press.

[9] Lok, S., Feiner, S. 2001. A Survey of Automated Layout Techniques for Information Presentations. Dept. of Computer Science, Columbia University

[10] Mardle, S. & Pascoe, S. An overview of genetic algorithms for the solution of optimisation problems. *University of Portsmouth*. Vol 13, Issue 1, 1999. Url: http://www.economicsnetwork.ac.uk/cheer/ch13_1/ch13_1p16.htm

[11] Moore, M. 2001. Teaching students to use genetic algorithms to solve optimization problems. In *Proceedings of the Seventh Annual Consortium For Computing in Small Colleges Central Plains Conference* on the *Journal of Computing in Small Colleges* (Branson, Missouri, United States). J. G. Meinke, Ed. Consortium for Computing Sciences in Colleges. Consortium for Computing Sciences in Colleges, 19-25.

[12] W3C – The World Wide Web Consortium – Url: www.w3.org.

[13] Yoshida, T., Hiroyasu, T., Miki, M., Ogura, M., Okamoto, Y.: *Energy Minimization of Protein Tertiary Structure by Parallel Simulated Annealing using Genetic Crossover*. Proceedings of 2002 Genetic and Evolutionary Computation Conference (GECCO 2002) Workshop Program (2002) 49-51

Appendix A: Guidance for prototype test participants

(This version is in Norwegian and has been slightly shortened to avoid duplicate images with the thesis)

Veiledning for testere.
Del av masteroppgave innen optimalisering av layout ved Høgskolen i Gjøvik.

Poenget med denne testen er at du som er kjent med bruk av publiseringsløsninger for web skal gjøre deg raskt kjent med prototypen av en nettavis som genererer forsidelayout for hver artikkel som publiseres. Prototypen presenterer en teknikk som bruker en algoritme for å genere layout basert på hvilke nyheter som skal vises for bruker. Du vil poste en nyhet med tilhørende bilde for så å gå til en forside-generator hvor du skal klikke deg frem til en layout du mener er bra nok til å vises for leserne. *Forventet tid brukt på testen er 30min.*

Gå gjennom følgende liste en gang, og svar så på noen spørsmål når du har utført alle punktene.

1. Gå til **prototype.draape.no** og logg inn med tildelt passord og brukernavn.
2. Trykk på ”Gå til nettavis” oppe i høyre hjørne og gjør deg raskt kjent med forsiden. (Artiklene er ikke lesbare utover selve forside-presentasjonen). Det vil ta 1,5 sek for artiklene å plassere seg (midlertidig).
   INFO: Forsiden er et lagret bestemt layout som admin har bestemt at bruker skal møte på forsiden. Under kategoriene (oppe til venstre) er ikke layouten forhåndslagret, men genereres når siden lastes. Det er i denne testen bare nok artikler under kategorien ”Kultur” til å teste dette. Gå til denne kategorien og trykke ”Refresh” i nettleseren for å se hvordan artiklene flytter på seg. Etter hvert som det postes artikler til forsiden som tilhører en kategori, vil kategori-sidene tilby dynamisk layout automatisk.
3. Gå tilbake til kontrollpanelet ved å trykke på tilbake-knappen i nettleseren.
4. Opprett en ny artikkel ved å trykke på linken ”Legg til nyhet” oppe til ventre.
5. Skriv inn overskrift, ingress og innhold og trykk på ’lagre’ nederst på siden. (Teksten i feltet ’Innhold’ vil ikke synes for leser i prototype, så ikke bruk tid på å legge inn noe her)
6. Etter at artikkelen er lagret kan du gå til ”Bildebehandling” øverst på siden og laste opp ønsket bilde til artikkelen. Velg plassering ”forside” eller ”innhold og forside” dersom du vil at bilde skal vises på forsiden i artikkelpresentasjonen.
7. Trykk på ”Rediger nyhet” for å komme tilbake til artikkelen.
8. Sett ønskede innstillinger for artikkel i listene øverst. Disse innstillingene medfører hvordan artikkelen blir presentert på forsiden av nettavisen.
   Beskrivelse for innstillinger:
   Aktiv: Velg om artikkel skal vises eller ikke.
   Kategori: Velg hvilken kategori artikkelen skal legges i.
   Bildeplassering: Velg hvor artikkelbilde skal ligge inne i presentasjonsboksen for gitt artikkel.
   Publiser i: Skal artikkelen bare vises på hovedside og i kategori velger du ”hovedside”. Valget ”kat.” medfører visning bare i kategorivisning.
   Kolonne span: Hvor mange kolonner (1-3) skal artikkelen være. Bare tilgjengelig når ”Type” er satt til Flytartikkel eller Prioritert.
   Type: Velg type for artikkel. Hovednyhet vises stort øverst. Flytartikkel plasseres der den passer best. Prioritert settes i den grad det lar seg gjøre høyt

på siden. Kan også sette et nummer for å velge rekkefølge på de prioriterte artiklene.

9. Lagre artikkel, og gå til ”Oversikt artikler” i menyen øverst.
10. Trykk på ”Generer forside” øverst på siden for nå å oppdatere layouten for forsiden, og for å kunne trykke deg frem til et godkjent layout.
    ➔ I tillegg til et vindu av selve nettavisen kommer det opp et lite vindu med oversikt over hvordan kjøringen gikk, og knapp for å kjøre igjen (”Execute”). Trykk på denne helt til artiklene ligger slik du ønsker, og du er fornøyd med layouten.

    OBS! Prototypen har en del postede artikler allerede. Her vil det f.eks. være en hovednyhet som alltid vil ligge øverst. Dersom du ønsker at din artikkel skal komme øverst, må du endre artikkeltype for den nåværende hovednyheten og sette artikkeltype ”hovednyhet” på den artikkelen du vil vise øverst.
11. Når du er fornøyd med layouten som vises, trykker du på ”Save solution”. Da lagres layoutet, og vil fra nå vises for brukere som besøker nettavisen.
12. For å se hvordan siden vil se ut for bruker etter at du har generert en layout, kan du gå tilbake til Cpanel ved å trykke på linken i headeren. Trykk deretter på linken oppe i høyre hjørne, ”Gå til nettavis”.

Nå kan du gå til **spørreundersøkelsen** og logge inn med tildelt brukernavn og passord for å svare på noen spørsmål angående prototypen. Trykk på ’Finish’ nederst til høyre når du har svart på alle spørsmålene.

Dersom du opplevde noen bugs eller andre feil under testen er det fint om du sender en notat på dette til gjermund@brabrand.no.

Takk for at du tok deg tid til denne testen!

Appendix B: Questions for prototype test participants

Spørsmål til testdeltakere av prototype */Questions for prototype test participants*

1. Hva synes du om det å bygge nettsiders layout basert på nettstedets innhold (dynamisk innhold i dynamisk layout) fremfor å presentere nettstedets innhold i en fast layout (dynamisk innhold i ett statisk layout)?

   What do you think of the idea of building a web site layout based on its content (dynamic content in a dynamic layout) over presenting the web site content in a static layout?

2. Hvor mange ganger (anslagsvis) måtte du kjøre generatoren for å få den layouten du endte opp med å lagre?

   How many times (approximately) did you have to execute the generator in order to accomplish the layout you ended up storing?

3. Hvordan opplevde du denne prosessen med å "trykke deg frem" til en god layout?

   How was your experience with the process of clicking towards a suited/approved layout?

4. Synes du at generatoren kom opp med tilfredstillende layout (måtte du inngå et kompromiss mellom dine forventninger og de faktiske resultatene), og/ eller ville du foretrekke å lage layouten selv?

   Did the generator come up with a satisfying layout (did you compromise with your expectations and the actual results) or would you prefer to make the layout yourselves.

5. I denne prototypen ble et kontrollpanel presentert ved lasting av nyhetssiden slik at du kunne generere layouten som skal vises for bruker. Tror du det ville være hensiktsmessig å gi brukeren muligheten til å endre layouten selv? Utdyp.

   In this prototype a control panel is presented along with the newspaper web site. Do you think it would be advantageous to offer this opportunity to consumers of the newspaper?

6. Følte du deg låst eller på noen måte bundet til funksjonaliten? Forklar.

   Did you feel locked or in any other way bound to the functionality. Explain.

7. Savnet du noen muligheter og/eller synes du generatoren hadde noen mangler, og i så fall hvilke?

   Did the prototype have any lacks or opportunities you would have liked to see in a system like the one presented? Which?

8. Ser du for deg andre bruksområder til automatisert layout slik som presentert i prototypen? Forklar.

Can you think of any other areas for automated layout the way it is presented in the prototype? Explain.

9. Var det vanskelig å sette seg inn i hvordan bruken av prototypen fungerte sammenlignet med publiseringsløsningen(e) du er vant til å bruke?

   Was it hard to understand and use the prototype compared with the publishing system you are used to?

10. Savnet du det å kunne kontrollere/påvirke automatikken i layout-genereringen i noen grad? I så fall hva ville du ønsket å kontrollere/påvirke?

    Did you miss any interaction with the automation presented in the layout generating process? If yes, what would you like to affect/control?