MAXIMIZING REVENUE WITH ALLOCATION OF MULTIPLE ADVERTISEMENTS ON A WEB BANNER

V.A.G. Boskamp
305434vb@student.eur.nl

Supervised by Dr.ir. Flavius Frasincar
frasincar@ese.eur.nl

Co-reader: Dr. Adriana Gabor
gabor@ese.eur.nl

Bachelor Thesis of Economics & Informatics,
Econometric Institute, Erasmus School of Economics,
Erasmus University Rotterdam, the Netherlands

July 9, 2009
Abstract

This paper deals with the allocation of multiple advertisements on a Web banner. The purpose is to find a pattern generating algorithm that maximizes the revenue of the allocated advertisements. We set out an extensive simulation using standard banner sizes, primary and secondary sorting criteria for the set of advertisements, and four heuristic algorithms. The heuristic algorithms presented in this paper are the left justified algorithm, the orthogonal algorithm, the GRASP constructive algorithm, and the greedy stripping algorithm. First we investigate the process of finding the optimal solution using brute force search. Then we run two benchmarks, one to compare the heuristics with the optimal solution, and one to gain better insight in the performance of the heuristic algorithms. Finding a suitable pattern generating algorithm is a tradeoff between effectiveness and efficiency. Results indicate that allocating advertisements with the orthogonal and left justified algorithm is most effective. In contrast, allocating advertisements using the GRASP constructive and greedy stripping algorithm is most efficient. Furthermore, the best settings per algorithm for each banner size are presented.
Contents

List of Figures 4

List of Tables 5

1 Introduction 6

1.1 Problem description 7
1.2 Methodology 8
1.3 Literature review 9
1.4 Structure 10

2 Experimental Design 11

2.1 Simulation overview 11
2.2 Configuration 11
2.2.1 Size of the banner 12
2.2.2 Sorting of the advertisements 13
2.2.3 Other variables 14
2.3 Concluding remarks 15

3 Optimal Solution 16

3.1 Formal problem definition 16
3.2 Finding the optimal solution 19
3.3 Brute force search 20
3.4 Concluding remarks 20

4 Heuristics 22

4.1 Initialization 22
4.2 Left justified algorithm 23
4.3 Orthogonal algorithm 23
4.4 The GRASP constructive algorithm 26
4.5 Greedy stripping algorithm 27
4.6 Concluding remarks 29
List of Figures

1.1 Sample from “Million Dollar Homepage” . . . . . . . . . . . . . . 7
2.1 Simulation setup . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.2 Distribution of ad widths for $336 \times 280$ banner . . . . . . 15
3.1 Visualization for Eq. (3.3), (3.4), (3.5) . . . . . . . . . . . . . . 18
4.1 Cursor position in left justified algorithm . . . . . . . . . . . . 23
4.2 Cursor position in orthogonal algorithm . . . . . . . . . . . . . 26
4.3 Free rectangles in GRASP algorithm . . . . . . . . . . . . . . 27
4.4 Example of greedy stripping, sorting $A_{sub}$ by price per pixel . . . . . . 29
5.1 Advertisement details . . . . . . . . . . . . . . . . . . . . . . . 31
5.2 Optimal solution . . . . . . . . . . . . . . . . . . . . . . . . . . 32
List of Tables

2.1 Standard banner sizes ........................................... 13
3.1 Model parameters ................................................... 17
5.1 Best results for each algorithm .............................. 32
5.2 Five point summary of the profit per banner pixel per algorithm 33
5.3 Five point summary of the execution time per algorithm in seconds 34
5.4 Profit per pixel for all primary sortings for each algorithm . . . 35
5.5 Execution time in seconds for all primary sortings for each algorithm 37
5.6 Influence of secondary sort on Price/pixel Desc primary sort . . . 38
5.7 Most effective algorithm settings 728 x 90 .................. 39
5.8 Most effective algorithm settings 234 x 60 .................. 39
5.9 Most effective algorithm settings 125 x 125 .................. 40
5.10 Most effective algorithm settings 120 x 600 ................. 40
5.11 Most effective algorithm settings 336 x 280 ................ 40
Chapter 1

Introduction

With the Web usage still growing, Web advertising becomes a more dominant form of marketing every year. According to the Interactive Advertising Bureau, Web advertising revenues for 2008 are totaled $23.4 billion in the U.S. only [1].

A special form of Web advertising is pixel advertisement. Pixel advertisement is the presentation of several small advertisements on a larger, two-dimensional space. It originated in 2005 from the English student Alex Tew’s “Million Dollar Homepage” [2]. In order to make some money he came up with the idea to sell advertising space in a unique concept. The homepage holds a 1000 by 1000 pixel grid from which blocks of 10 by 10 pixels could be bought for 1 dollar per pixel. Buyers could place an image on their pixels and let the image link to their website. The pixel advertisement website, being the first of its kind, became a great success. Many copycats arose afterwards, but could not repeat the success. In [3], the success factors and weak points of the concept are identified. Original pixel advertisement websites, displaying a page full of advertisements, are useless after the “Million Dollar Homepage” as the copycats showed. However, applying the concept in a different way may still be interesting. In this thesis, a modified version of the pixel advertisement concept is presented. We apply the concept to Web banners. Representing 22 percent of the total Web advertisement revenue in 2008, banner advertisements remain a significant source of income for Web advertisers [1]. A sample from the “Million Dollar Homepage” in the shape of a banner is given in Figure 1.1.

In Section 1.1, we set forth our concept and present the research questions. Section 1.2 presents the methodology used in this research. An overview of related literature is given in Section 1.3. Finally, the structure of the thesis is outlined in Section 1.4.
1.1 Problem description

In our modified version of the concept, advertisers commit small Web advertisements, without specifying a location on the banner where the advertisement should be placed. In the original approach, advertisers choose their own pixels from the ones available and are sure of placement. Here, not every advertisement is placed. Each advertisement has a certain price per pixel, which can be obtained either by negotiations with the space-owner or just the value the advertiser is willing to pay for it. If an advertisement is placed, the advertiser has to pay the costs corresponding to the size of the advertisement. When we have a set of advertisements, we allocate them across the banner. An assumption we make is that we have more advertisements than would fit on the banner. The goal of the space-owner is to maximize revenue, so preferably advertisements from the set are allocated in a way that yields the most profit. As a consequence, some advertisements of the set may not be placed. We name this modified version of the pixel advertisement concept, multiple advertisement allocation.

There is still innovation possible in the field of internet marketing. The original pixel advertisement concept is already being applied to banners\textsuperscript{1} but not on a large scale. With our modifications we are more aiming at a platform for banner advertisement marketing like Google’s AdWords\textsuperscript{2}. Such a platform should take all accompanying tasks out of hands. Advertisers submit small advertisements, specify their target subjects, and have their ad placed in a subject-related pool. The banner space owners supply their banners to the platform, where advertisements from the pool are allocated on the banner. The space owners receive commission in return.

A great advantage of our concept, having a set of advertisements larger than would fit on the banner, is that there is much competition among the advertisers. In the end this will lead to higher prices per pixel. However, the small size of the advertisements placed on the banner will keep the prices of the advertisements relatively low in comparison with normal banners. This makes internet advertising more accessible for anyone.

\textsuperscript{1}for example on \url{http://www.bradfallon.com/}
\textsuperscript{2}\url{http://adwords.google.com/}
In this research we deal specifically with allocating small advertisements on a banner. The goal is to gain insight in the different solutions available for the allocation process. The main research question is: \textit{How to efficiently generate the optimal allocation pattern to maximize revenue for multiple advertisement allocation?}

The problem tackled in this work can be seen a variant of the well-known cutting and packing problems in literature. When following the typology in [4], our problem can be classified as a \textit{two-dimensional, single, orthogonal, knapsack problem}. In the knapsack problem, a strongly heterogeneous assortment of small items has to be allocated to a given set of large objects. The availability of the large objects is limited in a way that not all small items can be accommodated. The objective is to maximize the value of the accommodated items. The term \textit{single} means we have only one large object to place our advertisements in, namely the banner. In \textit{orthogonal} patterns, the edges of small items are set parallel to the edges of the large object. In non-orthogonal patterns, the angle is optional [5]. The problem is NP-hard [6], making it extremely time-consuming to find the optimal solution. Therefore, besides the optimal solution, we apply heuristics to find adequate solutions.

The general research question has been divided in subquestions with the aim of developing a better understanding of the research.

1. \textit{How to formally specify the multiple advertisement allocation problem?}

2. \textit{What is the most effective heuristic for multiple advertisement allocation?}

3. \textit{What is the most efficient heuristic for multiple advertisement allocation?}

4. \textit{What is the best tradeoff between effectiveness and efficiency for multiple advertisement allocation?}

\subsection*{1.2 Methodology}

The chosen methodology is a simulation study. We will implement several solutions for generating allocation patterns for multiple advertisement allocation. The implementations consist of an optimal solution for benchmark purposes and several heuristic algorithms. We will analyse the performance of the different solutions with respect to profit maximization and efficiency. The efficiency is important for practical application of the algorithms. The execution times of the algorithms should stay within reasonable boundaries for Web use (i.e. 30 seconds). A more detailed description of the simulation configuration is given in Section 2.1.
1.3 Literature review

Related work can be categorized in either pixel advertising, the ad placement problem, or cutting and packing problems.

Until now, very little literature is available on pixel advertisement. In [3], Web advertisement in general is discussed and the “Million Dollar Homepage” is analysed. It also proposes some improvements for the pixel advertisement concept. In [7], a heuristic approach for placing multiple advertisements on a banner is proposed. The paper deals with the same concept as presented in this research.

All other literature on the placement of Web advertisements has been focusing on the ad placement problem. It was introduced in [8] as a variant of the bin packing problem. The ad placement problem (APP) focuses on time scheduling (or space sharing) of advertisements on a banner. The banner has multiple time slots through which is cycled over time. Every slot has a different allocation pattern of advertisements. Furthermore, it is concerned only with the placement of one advertisement on a banner or some advertisements side-by-side, whereby the height of the advertisements is equal to the height of the banner. Note that this differs from the pixel advertisement and multiple advertisement allocation approach, where advertisements are placed in a two-dimensional way. Since the APP is the most popular related problem we will give a short overview of the field. In [8], a distinction is made between the offline and online scheduling of advertisements. In the offline problem, we have a predefined set of advertisements to be scheduled. In the online problem, we receive requests for placement sequentially and we have to decide whether to accept these without knowledge of future requests. In most of the related work both approaches are discussed.

Another distinction made concerns the MINSPACE and MAXSPACE problem. In the MINSPACE problem, we are given a number of time slots. Time slots represent different allocation patterns of the same banner. The banner cycles through these slots over time. Besides the time slots, we are given a set of advertisements. In this problem, the objective is to allocate all advertisements over the time slots, while minimizing the banner size. The width and height of the banner are the same for all time slots. The goal of the MINSPACE problem is to find these minimum banner width and height. In the MAXSPACE problem, we are given the banner dimensions and the number of time slots. We are also given a set of advertisements. However, the objective of the MAXSPACE problem is to place advertisements in such a way that the revenue is maximized. This means not all advertisements are allocated. The advertisements should be allocated optimally over the time slots, while maximizing revenue. The goal of
the MAXSPACE problem is to find this allocation pattern.

For both problems, several solutions are available using polynomial time approximation algorithms [9][10][11], Lagrangian decomposition [12][13], column generation [13] and a hybrid genetic algorithm [14]. Based on the previous classifications, our research deals with an offline and MAXSPACE problem.

In Section 1.1, we already mentioned the cutting and packing problems. These type of problems are encountered in many industries, thus becoming a popular subject for researchers. A nice overview of the field is given in [4], including a categorization of known problems. Our problem is categorised as a two-dimensional single orthogonal knapsack problem. Several variants of the knapsack problem exist, which differ in either the dimension used, the number and size of the large rectangles, or that have some additional constraints. Other cutting and packing problems closely related to our problem are the placement problem and cutting stock problem. The difference between the placement problem and the knapsack problem is that the former has a weakly heterogeneous set of items to be placed, while the latter is characterised by a strongly heterogeneous set of items. Since, in our problem, advertisers may submit small advertisements from any width and height in steps of 10 pixels, we have a strongly heterogeneous set of items. The cutting stock problem requires that the set of items is completely allocated, and is therefore focusing on input (space) minimisation instead of output (profit) maximization.

1.4 Structure

The rest of this thesis is organized as follows. In Chapter 2, a detailed description of the experimental design is given. The simulation configuration and accompanying variables are formulated. Chapter 3 discusses the process of finding the optimal solution and presents a brute force search implementation. In Chapter 4, we present heuristic algorithms that generate adequate allocation patterns. Chapter 5 shows the results of the simulation experiments. The performance of the different algorithms are compared with respect to several criteria. Finally, in Chapter 6 we make some concluding remarks and propose suggestions for future research.
Chapter 2

Experimental Design

In this chapter we present the experimental design of our simulations. First, we will distinguish two simulations in Section 2.1. Then, in Section 2.2 we will set out the configuration of our main experiment. The configuration parameters and other variables are discussed in separate subsections. Section 2.3 puts our experimental design into perspective.

2.1 Simulation overview

In this research, we study the performance of different algorithms for allocating small advertisements on a banner. The performance in our problem can be defined in two ways. On the one hand, it can be defined as maximizing the profit, thereby focusing on effectiveness. On the other hand, it can be defined as the execution time of the algorithms, thereby focusing on efficiency. We measure both definitions of performance by running simulations. First, we will run a small simulation to benchmark the optimal solution against the heuristic solutions, the optimal-vs-heuristics benchmark. This is done in a separate simulation because finding the optimal solution is extremely time-consuming. The process of finding the optimal solution is discussed in Chapter 3. The settings for the optimal-vs-heuristics benchmark are given with the simulation’s result in Section 5.1. Second, we will run a large simulation to benchmark only the heuristic algorithms against each other, the heuristic-vs-heuristics benchmark. The rest of this chapter presents the experimental design for the latter simulation.

2.2 Configuration

In the heuristic-vs-heuristics benchmark simulation, every simulation cycle has a different configuration of its parameters. We distinguish the following three
configuration parameters. In every simulation cycle, one of these parameters is changed.

- Size of the banner
- Sorting of the advertisements
- Algorithm for ads placement

All possible combinations of values from these parameters are simulated. In this way we obtain unbiased results, from which conclusions are drawn. When running simulations with multiple configuration parameters it is necessary to specify the setup of the simulation. This defines which parameters stay fixed when another parameter is changed. The setup of our simulation is showed in Figure 2.1.

The most outer loop contains the banner size parameter. The inner loop contains the sorting criteria. For every banner size, we simulate all sorting criteria. Furthermore, for every sorting criteria we run all algorithms. During each cycle of the simulation the configuration parameters are registered. For each cycle the waste rate (ratio of unallocated space in the banner) and the total profit of the generated allocation pattern are calculated. The execution time and the number of advertisements placed are also registered. The experiment is implemented in Matlab R2008b and run on a Intel Core 2 Quad CPU at 2.40 GHz.

We will discuss our configuration parameters and other simulation variables in separate subsections. The algorithms are presented in Chapter 4.

### 2.2.1 Size of the banner

Five standard banner sizes commonly used in Web advertising [15], have been selected to be used for each of the simulation cycles. The width $W$ and height $H$ of the banners are shown in Table 2.1.

During the simulation the widths and the heights of the banners are also reverted to avoid bias towards a particular shape of the banner. In total this amounts to 9 different banners, since the square banner is not reverted.

![Figure 2.1: Simulation setup](image-url)
Table 2.1: Standard banner sizes

<table>
<thead>
<tr>
<th>W × H</th>
<th>Banner</th>
</tr>
</thead>
<tbody>
<tr>
<td>728 × 90</td>
<td>Leader board</td>
</tr>
<tr>
<td>234 × 60</td>
<td>Half banner</td>
</tr>
<tr>
<td>125 × 125</td>
<td>Square button</td>
</tr>
<tr>
<td>120 × 600</td>
<td>Skyscraper</td>
</tr>
<tr>
<td>336 × 280</td>
<td>Large rectangle</td>
</tr>
</tbody>
</table>

2.2.2 Sorting of the advertisements

For every allocation process we have a set of advertisements. During allocation, the algorithms iterate through this set sequentially. Therefore, the order of the set influences the generated pattern. We created the following sorting criteria from advertisement parameters. The advertisements can be ordered in both ascending and descending order.

- price per advertisement pixel \(p\)
- width \(w\)
- height \(h\)
- total area \(w × h\)
- flatness \(w/h\)
- proportionality \(|\log(w/h)|\)

The flatness specifies whether the advertisement is horizontally flat \((\text{width} > \text{height})\) or vertically flat \((\text{height} > \text{width})\). When the flatness is larger than 1 the rectangle is horizontally flat. When the flatness is smaller than 1 it is vertically flat. The proportionality refers to how much the rectangle resembles a square. A value of 0 for this attribute means that the rectangle is a square \((\log(1) = 0)\). Any higher value signifies that the rectangle is flat or tall. The difference between the proportionality and the flatness is that with the latter you can specify the shape preferred.

Normally, we would say that sorting to a descending price per pixel would give the most benefit. However, this is not always the case. The first advertisements that are allocated influence the continuation of the pattern. While the first advertisements may yield high revenues until that point, the next advertisements in the ordered set may not be placed due to space constraints.

\(^1\)Since the variable is named flatness we specified height > width as vertically flat, however in natural language tall is the best description.
Advertisements from the end of the ordered set may fill up the rest of the banner. In the end, the total revenue may be lower using this sorting criteria as with a different one. When ordering the advertisements to an ascending total area, many advertisements are placed and the waste rate is low. This may outperform other sortings with higher waste rates, but it also depends on the banner size.

When sorting the advertisements according to certain criteria, some advertisements may have the same value for that criteria. To achieve a more specific sorting we add a secondary sort, using one of the remaining criteria. Altogether the set of advertisements is sorted in \( \frac{12!}{10!} - 12 = 120 \) different ways, with two directions (ascending and descending) and excluding the situations where the primary sort equals the secondary sort.

### 2.2.3 Other variables

Besides the configuration parameters discussed, our problem has some other variables as well. Those variables include the set of advertisements and the banner price. For our simulation the advertisements are pseudo-randomly generated. The set of advertisements is semi-fixed during our simulation. Since the size of the advertisements is related to the size of the banner, we change it for every banner size.

**Size of the advertisements** In order to strike the right balance between randomness and using a realistic dataset for allocation during the simulation, we used a specific formula to generate the width and height of the ads. The most important constraint was that on the “Million Dollar Homepage” all pixels were sold to advertisers in blocks of 10 \( \times \) 10 pixels. Also, the dimensions of the pixel advertisements that were displayed on this webpage were not uniformly distributed but leaned towards a normal distribution with a mean equal to the minimal ad size of 10 \( \times \) 10 pixels. Naturally, we also do not want any ads that exceed the dimensions of the banner itself. For the generation of the width and height of the ads, we used the formula in Eq (2.1).

\[
w_i, h_i = \max \left( 10, \min \left( \min (W, H), \left\lceil \min (W, H) / 40 \times |\text{randn}| \right\rceil \times 10 \right) \right) \quad (2.1)
\]

To illustrate the distribution this formula creates, a histogram with the widths of a set of 500 ads for a 336 \( \times \) 280 banner is displayed in Figure 2.2. The heights of the ads in this particular example are distributed in the same manner. The set of 500 ads is an example, note that for our simulations we only create between 1.5 and 2 times as much advertisements as will be placed to avoid too many ties on the primary sorting.
Price of the advertisements  Another variable in our simulation is the price of the advertisements. We measure the price for advertisements in price per pixel. This price can differ between advertisements due to negotiations with the owner of the banner. Since we work with price per pixel, the prices of the advertisements are proportional to their dimensions. The larger the total area of the advertisement, the higher the revenue for the owner of the banner. The price per pixel is set to 10 with a random one decimal value between $-1$ and 1 added to this value, resulting in a uniform distribution between 9 and 11. The price of the advertisement is calculated by multiplying this price per pixel with its total area.

Price of the banner  We assume the banner itself has a certain value. Just like the price of the advertisements, the price of the banner is specified in price per banner pixel. In our simulations, the price of the banner is set to 4 per pixel. We assign this value to all unallocated pixels. The idea is that the unallocated pixels will be filled by the main sponsor for a lower price \cite{7}. The value assigned to the unallocated pixels is low enough to prefer the placement of advertisements.

2.3 Concluding remarks

We acknowledge the experimental design as presented in this chapter greatly influences the results of our simulation. Especially the set of advertisements has great influence. We think this design creates a set of ads from which general conclusions can be drawn.
Chapter 3

Optimal Solution

This chapter deals with the optimal solution for our problem. First, in Section 3.1 we present the formal problem definition. Then we discuss the process of finding the optimal solution and the tools available in Section 3.2. An algorithm for finding the optimal solution is presented in Section 3.3. Finally, we make some concluding remarks in Section 3.4.

3.1 Formal problem definition

Before searching for solutions we first have to define the problem in a formal mathematical way. We have a set \( A \) with \( n \) advertisements to allocate in banner \( B \). We assume we have more ads in \( A \) than would fit on the banner, thus not every advertisement is placed. Each advertisement \( a_i \in A \) has a width \( w_i \), height \( h_i \) and a price per pixel \( p_i \), with \( i \in \{1, \ldots, |A|\} \). Banner \( B \) has width \( W \) and height \( H \). The advertisements from \( A \) should be allocated on banner \( B \) so that the total value of the set of allocated advertisements \( A' \) is maximized. Each advertisement \( a_i \) in \( A' \) has a coordinate \((x, y)\) on banner \( B \), starting from the topleft corner of banner \( B \) at \((0,0)\). The value of an allocated advertisement in \( A' \) is defined by \( v_i \). Our objective is to maximize the total value of allocated ads in \( A' \). An overview of the model parameters is given in Table 3.1.

We can formulate the problem as a zero-one integer programming problem as described in [5]. Let

\[
X_i = \{x | 0 \leq x \leq W - w_i\}, \forall i \in \{1, \ldots, |A|\}
\]  

be the set of all possible points \( x \) along the width of banner \( B \) such that any ad \( a_i \) from \( A \) can be placed on banner \( B \) with its topleft at any \( x \in X_i \). Similarly
Table 3.1: Model parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{A}$</td>
<td>Set of advertisements</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of advertisements</td>
</tr>
<tr>
<td>$a_i$</td>
<td>Advertisement $i$ from $\mathcal{A}$</td>
</tr>
<tr>
<td>$w_i$</td>
<td>Width of advertisement $a_i$</td>
</tr>
<tr>
<td>$h_i$</td>
<td>Height of advertisement $a_i$</td>
</tr>
<tr>
<td>$p_i$</td>
<td>Price per pixel of advertisement $a_i$</td>
</tr>
<tr>
<td>$\mathcal{A}'$</td>
<td>Set of allocated advertisements</td>
</tr>
<tr>
<td>$v_i$</td>
<td>Value of allocated advertisement in $\mathcal{A}'$</td>
</tr>
<tr>
<td>$B$</td>
<td>Banner</td>
</tr>
<tr>
<td>$W$</td>
<td>Width of the banner</td>
</tr>
<tr>
<td>$H$</td>
<td>Height of the banner</td>
</tr>
<tr>
<td>$x$</td>
<td>X-coordinate on $B$</td>
</tr>
<tr>
<td>$y$</td>
<td>Y-coordinate on $B$</td>
</tr>
</tbody>
</table>

we define

$$\mathcal{Y}_i = \{y | 0 \leq y \leq H - h_i \}, \ \forall \ i \in \{1, \ldots, |\mathcal{A}|\} \quad (3.2)$$

as the set of all possible points along the height of banner $B$. Also, let (see Figure 3.1)

$$x_{ip} = \begin{cases} 
1 & \text{if } a_i \text{ is assigned to banner } B \text{ at position } p \text{ where } p \in \{1, \ldots, |\mathcal{X}_i|\} \\
0 & \text{otherwise} 
\end{cases} \quad (3.3)$$

$$y_{iq} = \begin{cases} 
1 & \text{if } a_i \text{ is assigned to banner } B \text{ at position } q \text{ where } q \in \{1, \ldots, |\mathcal{Y}_i|\} \\
0 & \text{otherwise} 
\end{cases} \quad (3.4)$$

$$z_{rs} = \begin{cases} 
1 & \text{if point } (r, s) \text{ in banner } B \text{ where } r \in \{1, \ldots, W - 1\} \\
& \text{and } s \in \{1, \ldots, H - 1\} \text{ has unallocated space} \\
0 & \text{otherwise} 
\end{cases} \quad (3.5)$$

Note that the possibility of replicates has been removed from the original formulation in [5], since our problem assumes every advertisement can be allocated
only once. Now, the integer programming formulation can be stated as follows:

\[
\text{max } \sum_{i=1}^{|A|} v_i \sum_{p=1}^{|X_i|} x_{ip} \tag{3.6}
\]

subject to

\[
q + h_i - 1 + w_i - 1 \leq x_{ip} - y_{iq} \leq w_i h_i
\]

\[
\forall i \in \{1, \ldots, |A|\}, \forall p \in \{1, \ldots, |X_i|\}, \forall q \in \{1, \ldots, |Y_i|\} \tag{3.7}
\]

\[
\sum_{p=1}^{|X_i|} x_{ip} \leq 1, \forall i \in \{1, \ldots, |A|\} \tag{3.8}
\]

\[
\sum_{p=1}^{|X_i|} x_{ip} = \sum_{q=1}^{|Y_i|} y_{iq}, \forall i \in \{1, \ldots, |A|\} \tag{3.9}
\]

In Eq. (3.6) the objective function maximizes the total value of the allocated advertisements. Constraint (3.7) ensures that at any pixel on the banner at most one ad is allocated and that whenever an ad is allocated all the correct pixels on the banner are marked as such by \(z_{rs}\). Note from Eq. (3.5) that \(z_{rs}\) is 0 when it is an allocated pixel. Constraints (3.8) and (3.9) ensures that any ad is allocated at most once on the whole banner.
\[
\sum_{i=1}^{N_A} h_i \sum_{p=r-w_i-1}^{r} x_{ip} + \sum_{s=0}^{H-1} z_{rs} = H, \quad \forall \ r \in \{1, \ldots, W-1\}, \quad \forall \ p \in \{1, \ldots, |X_i|\}
\]

(3.10)

\[
\sum_{i=1}^{N_A} w_i \sum_{q=s-h_i-1}^{s} y_{iq} + \sum_{r=0}^{W-1} z_{rs} = W, \quad \forall \ s \in \{1, \ldots, H-1\}, \quad \forall \ q \in \{1, \ldots, |Y_i|\}
\]

(3.11)

Constraints (3.10) and (3.11) ensure the sum of the widths and heights of the allocated ads and the amount of unallocated pixels in both horizontal and vertical direction is always equal to the bannerwidth \( W \) and bannerheight \( H \) resp.

Finally, we define the binary variables in Eq. (3.12). The detailed description of the binary variables can be found in Eq. (3.3), Eq. (3.4) and Eq. (3.5).

\[
x_{ip}, y_{iq} \in \{0, 1\}, \quad \forall \ i \in \{1, \ldots, |A|\}, \quad \forall \ p \in X_i, \quad \forall \ q \in Y_i
\]

(3.12a)

\[
z_{rs} \in \{0, 1\}, \quad \forall \ r \in \{1, \ldots, W-1\}, \quad \forall \ s \in \{1, \ldots, H-1\}
\]

(3.12b)

3.2 Finding the optimal solution

We have seen that our problem can be best described by a two-dimensional, single, orthogonal, knapsack problem. The knapsack problem, just like other cutting and packing problems, is a combinatorial optimization problem. These kind of problems can be solved in many different ways. For example by integer programming. However, this is fairly difficult due to the large number of variables and constraints. Another example is using meta-heuristics like simulated annealing, genetic algorithms, and tabu search. However, using meta-heuristics we are not certain of finding the optimal solution. To obtain the optimal solution in a straightforward way, search algorithms are used. These algorithms investigate the search space, the set of all possible solutions to a problem, and look for the best one. Search algorithms can be either uninformed or informed. Uninformed search algorithms are the simple ones, they just try all solutions in the search space. Informed search algorithms are more intelligent. They use heuristics to apply knowledge about the structure of the search space, thereby trying to reduce the execution time of the algorithm. Whether this yields optimal or suboptimal solutions depends on the heuristic used.

The problem variables we will use for the heuristic-vs-heuristics benchmark
are not suitable for the optimal-vs-heuristics benchmark. Finding the optimal solution with such a problem size is extremely time-consuming. There are two ways to decrease the execution time. One is to use more computational power, possibly dividing the computations over several computers. Another one is to use an informed search algorithm. This will relatively speed up the process, but with our problem size this is still extremely time-consuming. Therefore, instead of trying to decrease the execution time, we just decrease the problem size. In the optimal-vs-heuristics benchmark simulation we use a very small banner and small advertisements. We choose for this approach because we still want to give an indication of the differences in effectiveness and efficiency for the different solution approaches. We will use a standard uninformed brute force search to obtain the optimal solution for this relatively simple problem. The algorithm is presented in the next section. The settings of the simulation variables are defined in Section 5.1.

3.3 Brute force search

Brute force search, also referred to as exhaustive search, is a problem-solving technique in which all possible candidate-solutions are checked in order to find the optimal solution. It is a search algorithm that uses no information other than the initial state, the operators of the space, and a test for a solution [16]. The algorithm is used in many problems, all with a slightly different implementation to fit the problem. Our recursive implementation is presented in Algorithm 1.

Before the algorithm starts, the parameters \( B, W, H, \) \( \text{col}, \) \( \text{row} \) and \( \text{maxprofit} \) have to be initialized. Parameters \( \text{col} \) and \( \text{row} \) are cursor-variables, corresponding to the column and row position in the banner when it is seen as a matrix. After the initialization the algorithm starts. On every location of the banner we try to place every advertisement once. The recursion makes sure we generate all possible patterns, using tree-traversal. Once an advertisement has been placed, we continue allocating the remaining advertisements on the rest of the banner, trying all possible combinations of advertisements and locations. Every new optimal allocation pattern is stored, eventually yielding one or more solutions which maximize profit.

3.4 Concluding remarks

In this chapter we answered the first research subquestion, how to formally specify the multiple advertisement allocation problem? Furthermore, we discussed the optimal solution. We have seen the optimal solution is usually found us-
Algorithm 1 *Brute force search* algorithm

This algorithm is recursive. Every recursion step keeps track of its own parameters. During a recursion call, parameters are given to the next level. Before the first cycle the parameters have to be initialized.

\[
\textbf{function} \quad \text{brute-force-search} (B, W, H, \text{col}, \text{row}, \text{maxprofit}) \\
\{ \\
\text{while} \; \text{col} \leq W \; \text{do} \\
\quad \text{while} \; \text{row} \leq H \; \text{do} \\
\quad \quad \text{if} \; \text{B(col,row)} \; \text{is free} \; \text{then} \\
\quad \quad \quad \text{i} := 1; \\
\quad \quad \quad \text{while} \; \text{i} \leq n \; \text{do} \\
\quad \quad \quad \quad \text{if} \; a_i \; \text{fits on this location} \; \text{then} \\
\quad \quad \quad \quad \quad \{\text{Place } a_i \; \text{on } B\} \\
\quad \quad \quad \quad \quad \{\text{Remove } a_i \; \text{from } \mathcal{A} \; \text{and add } a_i \; \text{to } \mathcal{A}'\} \\
\quad \quad \quad \quad \quad \{\text{Compute total profit of } \mathcal{B}\} \\
\quad \quad \quad \quad \quad \text{if} \; \text{profit} \geq \text{maxprofit} \; \text{then} \\
\quad \quad \quad \quad \quad \{\text{Store allocation pattern}\} \\
\quad \quad \quad \quad \quad \text{maxprofit} := \text{profit} \\
\quad \quad \quad \end{while} \\
\quad \quad \quad \text{brute-force-search} (B, W, H, \text{col}, \text{row}, \text{maxprofit}) \\
\quad \quad \quad \{\text{Remove advertisement } a_i \; \text{from } \mathcal{B}\} \\
\quad \quad \quad \{\text{Remove } a_i \; \text{from } \mathcal{A}' \; \text{and add } a_i \; \text{to } \mathcal{A}\} \\
\quad \quad \end{while} \\
\quad \text{row} := \text{row} + 1; \\
\end{while} \\
\quad \text{col} := \text{col} + 1; \\
\text{row} := 1; \\
\text{end while} \\
\} \\
\]

For our experiment we use *brute force search*. In every problem with the objective of maximizing revenue, one wants to find the best solution. However, finding the optimal solution can be very time-consuming. The costs associated to this process are not always worth the effort. With possible Web use in mind, we want the maximum execution times of the allocation process to be under 30 seconds. Efficient solutions are obtained by heuristic algorithms, which are presented in Chapter 4. The goal is to create heuristic algorithms that approach the effectiveness of the optimal solution as much as possible.
Chapter 4

Heuristics

In this chapter we present four heuristics for multiple advertisement allocation. First we give a small initialization algorithm for our heuristics in Section 4.1. Then, we present the left justified algorithm in Section 4.2. The orthogonal algorithm is presented in Section 4.3. The GRASP constructive algorithm is presented in Section 4.4. In Section 4.5, the greedy stripping algorithm is presented. Finally, some concluding remarks are made in Section 4.6.

4.1 Initialization

The initialization step considers the sorting of the set of advertisements $\mathcal{A}$, and is identical for all heuristic algorithms. First the values of the primary and secondary sort, $s_1$ and $s_2$ are checked. Their values correspond to the attributes described in Section 2.2.2 and may be either positive or negative corresponding to an ascending and descending sorting order. The primary and secondary sort may not apply to the same criteria, and we also avoid duplicate sorting in opposite direction. Sorting $\mathcal{A}$ according to $s_1$ and $s_2$ yields $\mathcal{A}_0$. This is the ordered set of advertisements through which we iterate with $i$ in each algorithm. Furthermore, we initialize $n$, the number of advertisements in $\mathcal{A}_0$. The initialization step is given in Algorithm 2.

Algorithm 2 Heuristic algorithm initialization

Ensure: $s_1 \neq s_2 \& s_1 \neq -s_2$ \{Avoid duplicate sorting and duplicate sorting in opposite direction\}
Sort all $a_i$ in $\mathcal{A}$ first by $s_1$ and then by $s_2$
$\mathcal{A}_0$ \{Ordered set $\mathcal{A}$\}
$i := 1$ \{Iterator for $\mathcal{A}_0$\}
n \{Number of advertisements in $\mathcal{A}_0$\}
4.2 Left justified algorithm

The \textit{left justified algorithm} iterates through the ordered set of advertisements \( A_0 \). For each advertisement \( a_i \) it scans through the columns of banner \( B \) from top to bottom. If the end of a column is reached the iterator continues at the next column on the first row, and so on. In Figure 4.1 the cursor position is visualized for the first few iterations. When an available field is found and \( a_i \) fits on the empty location, it is placed in the banner. Advertisements are placed with the top left corner at the cursor position. If \( a_i \) goes horizontally out of bounds for a specific cursor position, we are unable to get it allocated and continue with the next advertisement. In contrast, if \( a_i \) goes vertically out of bounds we move the cursor to the first row of the next column. When we have iterated through all advertisements from ordered set \( A_0 \), the algorithm stops and the allocation pattern is returned. The details of this algorithm are shown in Algorithm 3.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig41}
\caption{Cursor position in left justified algorithm}
\end{figure}

4.3 Orthogonal algorithm

The \textit{orthogonal algorithm} iterates through the ordered set of advertisements \( A_0 \) and places advertisements as close to the top left corner as possible. The algorithm looks for free locations for the current advertisement by moving diagonally from the top left corner \((r,c) = (1,1)\) of banner \( B \). At each step, the algorithm searches at location \((r,i)\) with \( i \in \{1\ldots c\} \) and \((i,c)\) with \( i \in \{1\ldots r\} \) for the first free space where the advertisement can be allocated. We store the first free location from the borders to \((r,c)\) in variables \textit{verticalplace} and \textit{horizontalplace}. After that we compare them and allocate \( a_i \) on the location closest to the top left corner\footnote{This is computed by taking the sum of the \( x \) and \( y \) coordinates of both locations. The summation that yields the smallest outcome is the location closest to the top left corner.}. When there is a tie we choose the one on the vertical search path. When \( a_i \) is allocated, we start again in the topleft corner of \( B \), trying to allocate \( a_{i+1} \).

When we fail to allocate an advertisement for a certain \((r,c)\) we continue to walk diagonally down-right by increasing both \( r \) and \( c \) by one. When the final
**Algorithm 3** Left justified algorithm

Run heuristic algorithm initialization (Algorithm 2)

for i = 1 to n do
  Select a_i from A_0
  finished := false
  r := 1 {Current row in B}
  c := 1 {Current column in B}
  while finished = false do
    if a_i fits on B_r,c then
      {Allocate a_i on B_r,c}
      for p = c to c + x_i do
        for q = r to r + y_i do
          B_{p,q} := i
        end for
      end for
      finished := true
    else if r + y_i > \mathcal{H} then
      if c < W then
        c := c + 1
        r := 1
      else
        finished := true
      end if
    else if r + x_i > W then
      finished := true
    else
      if r < \mathcal{H} then
        r := r + 1
      else
        if c < \mathcal{W} then
          c := c + 1
          r := 1
        else
          finished := true
        end if
      end if
    end if
  end while
end for

return B

row is reached, but there are still columns left, we only increase the column. When the final column is reached, but there are still rows left, we only increase the row. This means that after we start walking diagonally, we will eventually switch to walking either right or down, except for the situation in which banner B is a square.

In Figure 4.2 the cursor positions for the orthogonal algorithm are given. The arrows indicate the directions we take into account when searching for the location closest to the top left corner. In the most right cursor position in Figure 4.2 we can see that the algorithm skips searching horizontally to the left when the final row is reached. This is because the banner is flat \((W/\mathcal{H} > 1)\) and we already searched that way before. An analogue reasoning goes for tall banners.
Orthogonal algorithm

Run heuristic algorithm initialization (Algorithm 2)

for $i = 1$ to $n$

Select $a_i$ from $A_0$

$\textit{finished} := \text{false}$ \{To stop searching locations for this $a_i$\}

$r := 1$ \{Current row in $B$\}

c := 1 \{Current column in $B$\}

$\textit{rowscompleted} := \text{false}$, $\textit{colscmpleted} := \text{false}$

verticalfound := false, horizontalfound := false

verticalplace := (0, 0), horizontalplace := (0, 0) \{To store candidate locations\}

colposition := 0, rowposition := 0 \{To store chosen location\}

While $\textit{finished} = \text{false}$ and ($\textit{rowscompleted} && \textit{colscompleted}$) = false do

if $\textit{colscomplete} = \text{false}$ then

{Search column from top border to cursor for candidate location}

for $p = 1$ to $r$ do

if $a_i$ fits on $B[c,p]$ then

store $(c, p)$ in verticalplace, verticalfound := true, break;

end if

end for

end if

if $\textit{rowscomplete} = \text{false}$ then

{Search row from left border to cursor for candidate location}

for $q = 1$ to $c$ do

if $a_i$ fits on $B[q,r]$ then

store $(q, r)$ in horizontalplace, horizontalfound := true, break;

end if

end for

end if

if verticalfound = true or horizontalfound = true then

if verticalfound = true && horizontalfound = true then

{Select location closest to left or upper border}

if $c + p$ (verticalplace) $<=$ $q + r$ (horizontalplace) then

$\textit{colposition} := c$, $\textit{rowposition} := p$

else

$\textit{colposition} := q$, $\textit{rowposition} := r$

end if

else if verticalfound = true then

$\textit{colposition} := c$, $\textit{rowposition} := p$

else if horizontalfound = true then

$\textit{colposition} := q$, $\textit{rowposition} := r$

end if

end if

{Allocate $a_i$ on $B[\textit{colposition, rowposition}]$}

for $f = \textit{colposition}$ to $\textit{colposition} + x_i$, do

for $g = \textit{rowposition}$ to $\textit{rowposition} + y_i$ do

$B[f,g] := i$

end for

end for

$\textit{finished} := \text{true}$

end while

if $r < H$ then

$r := r + 1$

else

$\textit{rowscompleted} := \text{true}$

end if

if $c < W$ then

c := c + 1

else

$\textit{colscompleted} := \text{true}$

end if

end for

return $B$
When the final row and column are reached and we still failed to allocate advertisement $a_i$, we start again in the top left corner of the banner and try to allocate the next advertisement from $A_0$. When we iterated through all advertisements in $A_0$, the algorithm stops. The details of this algorithm are shown in Algorithm 4.

![Figure 4.2: Cursor position in orthogonal algorithm](image)

4.4 The GRASP constructive algorithm

The GRASP constructive algorithm is based on the greedy randomized adaptive search procedure (GRASP) for the constrained two-dimensional non-guillotine cutting problem [17]. In GRASP, an iterative procedure combines a constructive phase and an improvement phase. In the constructive phase, a solution is build using a greedy heuristic. In the improvement phase, a local search procedure tries to improve the solution. The algorithm was originally produced for the cutting stock problem, but with some modifications it fits our problem as well.

It has a different approach than the other algorithms discussed. The only thing they have in common is the initialization with the sorting of advertisements. Sorting the advertisements to certain criteria makes the algorithm greedy. The main difference with the other algorithms is that we don’t search the banner for free space. Rather, we store rectangles of free space in set $L$. Free rectangles are parts of the banner where no advertisement is allocated yet. Initially, set $L$ contains only the full banner. To allocate advertisements, the following procedure is followed.

First, we take the smallest rectangle of $L$ in which an advertisement from list $A_0$ can fit. Then, we place an advertisement $a_i$ from ordered set $A_0$ that fits in the free rectangle. Whenever an advertisement is placed in a rectangle, new free rectangles are formed and added to $L$, while the original rectangle is removed from $L$. We always place the advertisement in a corner of the rectangle which is closest to a corner of the banner, and cut the free space left in such a way that it yields optimal new free rectangles. In Figure 4.3, the free rectangles 1, 2, and 3 are formed by placing an advertisement. In order to obtain the optimal new free rectangles we merge either rectangles 1 and 2, or 2 and 3. We choose the
merge for which the largest rectangle can accommodate the best advertisement from \( A_0 \). If there is a tie, we just choose the merge which yields a new free rectangle with the largest area. Whenever we fail to allocate any advertisement from \( A_0 \) in a rectangle from \( L \), we mark the rectangle as \textit{used}. This allows us to eventually stop the algorithm.

Figure 4.3: Free rectangles in GRASP algorithm

The GRASP constructive algorithm differs from the original GRASP approach, since we integrate the constructive phase and improvement phase. To enlarge the chance for free rectangles to be allocated by an advertisement, we directly merge adjacent free rectangles after they appear in \( L \). The new larger free rectangle is added to \( L \) and the merged rectangles are removed. In the original approach, this is only done in the improvement phase. This means you settle for a weak solution from the constructive phase, hoping for better results in the improvement phase. In the local search procedure of the improvement phase, some \textit{blocks} – in our case advertisements – are removed from the solution, remaining pieces are moved to the corners, and adjacent empty rectangles are merged. The new empty rectangles are filled again using the constructive phase. Only if the solution improved the move is made. Otherwise, some other blocks are removed from the solution, followed by the same procedure. For our application, merging empty rectangles on the go is much more efficient than introducing a local search procedure.

After we have processed a rectangle we continue with the next smallest rectangle from \( L \) not used before. When there are no free rectangles left (\( L \) is empty, the full banner is allocated) or no advertisements from list \( A_0 \) that fit any of the remaining rectangles, the algorithm stops. A brief overview of this algorithm is shown in Algorithm 5.

4.5 Greedy stripping algorithm

In the \textit{greedy stripping} algorithm, advertisements are allocated in \textit{strips}. These strips are filled from left to right or top to bottom – depending on the shape of the banner. If the banner is tall we create horizontal strips. If the banner is
Algorithm 5 \textit{GRASP constructive algorithm}

Run heuristic algorithm initialization (Algorithm 2)
\[ \text{\textit{finished}} := \text{false} \]  
(\text{Boolean variable to stop the algorithm})

\begin{algorithm}
\begin{algorithmic}
\State \( \mathcal{L}_j := \text{smallest rectangle from } \mathcal{L} \text{ not marked as } \text{used} \).
\State \( \text{ad}_{\text{placed}} := \text{false} \).
\While {\( \text{ad}_{\text{placed}} = \text{false} \) \&\& \( i \leq n \) }
\State \( \text{Loop through } \mathcal{A}_0 \text{ until we placed an advertisement or iterated through all ads} \).
\If {Advertisement \( a_i \) fits in rectangle \( \mathcal{L}_j \) }
\State \( \text{Place advertisement } a_i \text{ in banner } B \).
\State \( n := n - 1 \).
\EndIf
\If {\( \mathcal{L}_j \) is not completely filled by \( a_i \) }
\State \( \text{Cut new rectangle(s). If there three rectangles formed, choose the combination} \)
\State \( \text{of rectangles that can accommodate the best advertisement from } \mathcal{A}_0. \text{ If there is} \)
\State \( \text{a tie, choose the one yielding the highest total area} \).
\State \( \text{Add new free rectangles to set } \mathcal{L} \).
\State \( \text{Remove allocated rectangle } \mathcal{L}_j \text{ from set } \mathcal{L} \).
\Else
\State \( \mathcal{L}_j \) is completely filled by \( a_i \), no new rectangles are created.\)
\State \( \text{Remove allocated rectangle } \mathcal{L}_j \text{ from set } \mathcal{L} \).
\EndIf
\Else
\State \( i := i + 1 \);  \( \text{Go to next advertisement} \).
\EndIf
\EndWhile
\State \{Search for adjacent rectangles in set \( \mathcal{L} \). Merge them, add the new rectangle to \( \mathcal{L} \) and remove the merged rectangles from \( \mathcal{L} \}\}.
\State \{Mark \( \mathcal{L}_j \) as \text{used}, so we know we have already used this rectangle\}.
\State \{Now check whether we have already iterated through all rectangles in \( \mathcal{L} \)\}.
\If {all rectangles in \( \mathcal{L} \) are marked as \text{used} }
\State \( \text{finished} := \text{true}; \text{Stop algorithm} \).
\EndIf
\EndWhile
\Return \( B \).
\end{algorithmic}
\end{algorithm}

flat we create vertical strips. The first advertisement from \( \mathcal{A}_0 \) is placed, and its width – or height, depending on the shape of the banner – determines the width – or height – of the strip. Then we search \( \mathcal{A}_0 \) for advertisements that fit inside the strip and place them in a subset \( \mathcal{A}_{\text{sub}} \). This set contains \( n_{\text{sub}} \) advertisements. The subset is then ordered according to either (1) width or height – depending on the shape of the banner – descending, (2) price per pixel descending, or (3) following the primary sort of \( \mathcal{A}_0 \)\footnote{These criteria represent three implementations of the same algorithm. All are part of the simulations.}. After that we iterate through \( \mathcal{A}_{\text{sub}} \) trying to place the advertisements in the strip. Figure 4.4 displays an example of the greedy stripping algorithm in action. An advertisement \( a_c \) was placed on the first location of a tall banner. It directly limits the height of strip number 1 to
In this example advertisements from the subset are placed according to a descending price per pixel.

Whenever an advertisement $a_i$ from $A_{sub}$ is allocated, we remove it from original set $A_0$. If we reached the end of the strip or the subset is empty we create a new strip. Since $A_0$ has changed after filling a strip, we start the next strip with the first advertisement from $A_0$. We continue doing this until we iterated through all advertisements from $A_0$, then the algorithm stops.

The details of this algorithm are shown in Algorithm 6. We only show the case for which banner $B$ is tall. When $B$ is flat, an analogue procedure can be followed. The only difference is that the row/height-variables are exchanged with the col/width-variables.

\[ \begin{array}{c}
\text{Figure 4.4: Example of greedy stripping, sorting } A_{sub} \text{ by price per pixel} \\
\end{array} \]

### 4.6 Concluding remarks

In this chapter we presented four heuristic algorithms for multiple advertisement allocation. We think the left justified, orthogonal, GRASP constructive, and greedy stripping algorithm generate adequate allocation patterns. In Chapter 5 we will present the results of our simulations from which we get more insight in the algorithms’ performance. We are especially curious about the performance of the GRASP constructive algorithm with respect to both effectiveness and efficiency against the left justified and orthogonal algorithm, since their approach is completely different.
Algorithm 6 Greedy stripping algorithm

Run heuristic algorithm initialization (Algorithm 2)

\begin{align*}
r & := 1 \{ \text{Current row in } B \} \\
c & := 1 \{ \text{Current column in } B \} \\
\text{finished} & := false \\
\end{align*}

if \( W < H \) then

\begin{itemize}
  \item \( B \) is a tall banner, we create horizontal strips
\end{itemize}

next strip top location := 1 \{ Keeps track of next strip location \}

while \( \text{finished} = false \&\& A_0 \neq \emptyset \) do

\begin{align*}
\text{Get } a_i \text{ from } A_0 \\
\text{found ad} & := false \\
r & := \text{next strip top location} \\
\end{align*}

if next strip top location + \( h_i \) > \( H \) then

\begin{itemize}
  \item if \( i + 1 > n \) then \\
    \begin{align*}
    \text{finished} & := true, \text{ break; } \{ \text{All ads have been checked, stop algorithm} \} \\
    \end{align*}
      \item \( i := i + 1 \) \{ \text{Try next advertisement} \}
\end{itemize}

else

\begin{align*}
\text{found ad} & := true \{ \text{We start a new strip with this } a_i \} \\
\text{next strip top location} & := \text{next strip top location} + h_i \\
\end{align*}

end if

if \( \text{found ad} \) := true then

\begin{itemize}
  \item \{ Create subset \( A_{\text{sub}} \) of \( A_0 \) with advertisements that have same or lower \( h_i \) \}
  \item \{ Sort \( A_{\text{sub}} \) to one of the three criteria discussed in Section 4.5 \}
\end{itemize}

for \( s = 1 \) to \( n_{\text{sub}} \) do

\begin{itemize}
  \item \{ Get \( a_s \) from \( A_{\text{sub}} \) \}
  \item if \( a_s \) fits on \( B_r,c \) then \\
    \begin{itemize}
      \item \{ Place \( a_s \) on selected location \}
      \item \{ Remove \( a_s \) from \( A_0 \) \}
      \item \( n := n - 1 \) \\
      \item if \( c + w_s \leq W \) then \\
      \begin{align*}
      c & := c + w_s \{ \text{Next free location} \} \\
      \end{align*}
      \item else \\
      \begin{itemize}
        \item \( c := 1 \) \{ \text{Start with new strip} \}
        \item \( i := 1 \) \{ \text{\( A_0 \) has changed, start on top} \}
        \item break \{ \text{Break out of for-loop} \}
      \end{itemize}
    \end{itemize}
  \item \{ \text{\( A_0 \) has changed, start on top} \}
  \item end if
\end{itemize}

end for

\begin{align*}
\text{c} & := 1 \{ \text{We have iterated through all ads in } A_{\text{sub}}, \text{ start with new strip} \} \\
\end{align*}

end if

else

\begin{itemize}
  \item \( B \) is a flat banner, we create vertical strips. This part of the code is analogue to the part above, except now all width-variables are exchanged with height-variables
\end{itemize}

end if

return \( B \)
Chapter 5

Simulation Results

In this chapter we analyse the results of our simulations. We run two benchmark simulations. We compare the different algorithms with respect to profit and execution time. In addition, we draw conclusions based on the properties of the data set. In Section 5.1 we present the results of the optimal-vs-heuristics benchmark. Section 5.2 presents the results of the heuristic-vs-heuristics benchmark. Finally, in Section 5.3 we give our concluding remarks.

5.1 Optimal-vs-heuristics benchmark

This simulation benchmarks the optimal solution procedure, using brute force search, against the heuristics presented in this paper. The purpose of this benchmark is to gain insight in the effectiveness-efficiency tradeoff we discussed in Chapter 3. We have seen that finding the optimal solution is extremely time consuming. To overcome this problem we decrease the problem size for this particular benchmark. Instead of the banner sizes present in Table 2.1 we used only one small banner of $W = 6$ and $H = 4$. The benchmark was run with a set of 9 advertisements of which the details are shown in Figure 5.1.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$w_i$</th>
<th>$h_i$</th>
<th>$p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>9.50</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>10.00</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>10.50</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>9.20</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>9.70</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>10.30</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>10.80</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
<td>10.10</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>9.90</td>
</tr>
</tbody>
</table>

Figure 5.1: Advertisement details
There is a difference between the workings of the brute force search and the heuristics with respect to the sorting of the set of advertisements. The brute force search algorithm just tries all possible combinations, no sorting criteria are specified. The heuristics, in contrast, are assigned a primary ($s_1$) and secondary ($s_2$) sorting criteria. In this benchmark we tried all possible combinations of a set of sorting criteria for the heuristics.

Despite the decreased problem size, the brute force algorithm completed after almost two hours on an Intel Core 2 Quad CPU at 2.40 GHz. The optimal solution found is shown in Figure 5.2. Note that there are several optimal solutions possible by exchanging the allocated advertisements within the banner. The first optimal solution was found after 42 minutes, but the optimum was not known yet. It could have easily been found at the end of the brute force search. This emphasizes the difficulty in finding the optimal solution in a relatively small amount of time. Furthermore, increasing the problem size increases the execution time of finding the optimal solution exponentially. In Table 5.1, the best results for each algorithm are shown.

![Figure 5.2: Optimal solution](image)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>Profit</th>
<th>Exec. time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force search</td>
<td>n/a</td>
<td>n/a</td>
<td>243.00</td>
<td>7077.104768</td>
</tr>
<tr>
<td>Left justified</td>
<td>Total area Desc</td>
<td>Price/pixel Desc</td>
<td>243.00</td>
<td>0.000268</td>
</tr>
<tr>
<td>Left justified</td>
<td>Width Desc</td>
<td>Price/pixel Desc</td>
<td>243.00</td>
<td>0.000243</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Total area Desc</td>
<td>Price/pixel Desc</td>
<td>243.00</td>
<td>0.000503</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Width Desc</td>
<td>Price/pixel Desc</td>
<td>243.00</td>
<td>0.000572</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Shape Desc</td>
<td>Price/pixel Desc</td>
<td>243.00</td>
<td>0.001468</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Width Desc</td>
<td>Price/pixel Desc</td>
<td>243.00</td>
<td>0.001320</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Height Asc</td>
<td>Price/pixel Desc</td>
<td>243.00</td>
<td>0.001472</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Height Desc</td>
<td>All Desc the same</td>
<td>193.00</td>
<td>0.000474</td>
</tr>
<tr>
<td>Greedy stripping 2</td>
<td>Height Desc</td>
<td>All Desc the same</td>
<td>193.00</td>
<td>0.000479</td>
</tr>
<tr>
<td>Greedy stripping 3</td>
<td>Total area Desc</td>
<td>All the same</td>
<td>181.30</td>
<td>0.000528</td>
</tr>
<tr>
<td>Greedy stripping 3</td>
<td>Height Desc</td>
<td>All Desc the same</td>
<td>181.30</td>
<td>0.000509</td>
</tr>
</tbody>
</table>

From these figures we can conclude that it is much more efficient to use heuristics for multiple advertisement allocation. In addition, the effectiveness of some heuristics was equal to the brute force search. However, the problem size is too small to draw generally valid conclusions. Because of the small problem size, we have a larger chance of finding the optimum, especially with 120 different sortings. Nevertheless, the previous experiment – despite being performed on a

---

1. Note that Asc means ascending and Desc means descending.
small data set – gives us an idea of the results obtained with brute force search, and the differences with respect to the results obtained using heuristics.

5.2 Heuristic-vs-heuristics benchmark

In this simulation we benchmark the heuristics presented in this paper with respect to effectiveness and efficiency. The configuration of this benchmark was explained in Section 2.2. To summarize, the configuration parameters consisted of 9 different banner sizes, 120 different sortings of the set of advertisements, and 6 algorithms. Altogether this resulted in 6480 simulation cycles.

To avoid bias towards particular banner sizes, we normalize the profit and execution time. We define the profit per banner pixel $P_{\text{pixel}} = \frac{P_{\text{total}}}{W \times H}$, wherein $P_{\text{total}}$ is the total profit of the allocated pattern. Furthermore, we define the execution time per banner pixel $E_{\text{pixel}} = \frac{E_{\text{total}}}{W \times H}$, wherein $E_{\text{total}}$ is the total execution time for the allocated pattern. Since the same set of advertisements is used for all heuristic algorithms, we can evaluate their performance by comparing the normalized profits and execution times.

In Section 5.2.1 we discuss the overall performance of the different algorithms. Section 5.2.2 focuses on the sorting of the set of advertisements. More specific results for each banner size are presented in Section 5.2.3. Finally, our concluding remarks are made in Section 5.3.

5.2.1 Overall performance

In this section we present some statistics about the overall performance of the algorithms. We consider the full data set that was obtained, without specifying any banner size or sorting criteria. The profits per banner pixel are aggregated – average for different banner sizes and sorting criteria – per algorithm. Note that we also reverted the width and height of the banner to avoid bias towards a particular shape of the banner. In this way we reach a general conclusion about which algorithm to prefer for multiple advertisement allocation. A five point summary of the distribution of $P_{\text{pixel}}$ for each algorithm is shown in Table 5.2.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Minimum</th>
<th>Q1</th>
<th>Median</th>
<th>Mean</th>
<th>Q3</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRASP con.</td>
<td>7.298</td>
<td>9.120</td>
<td>9.632</td>
<td>9.486</td>
<td>9.890</td>
<td>10.670</td>
</tr>
<tr>
<td>Greedy str. 2</td>
<td>5.419</td>
<td>7.043</td>
<td>8.213</td>
<td>8.127</td>
<td>9.369</td>
<td>10.290</td>
</tr>
</tbody>
</table>

Table 5.2: Five point summary of the profit per banner pixel per algorithm

33
We can note that overall, the distribution of the orthogonal algorithm is around the highest values for the price per pixel. The distribution of the left justified algorithm is even narrower, it starts at the same minimum but ends at a lower maximum. The GRASP constructive algorithm and the greedy stripping algorithms have a wider distribution. Notice that the maximum value of the price per pixel distribution for the GRASP constructive algorithm equals the maximum value obtained by the orthogonal algorithm. The greedy stripping algorithms are the most ineffective. We have run 3 different implementations of the greedy stripping algorithm, each one representing a different sorting of the subset. Greedy stripping 1 is most effective. This can be explained by the sorting of the subset, which is sorted based on a descending width or height – depending on the shape of the banner. In the end this yields a lower waste rate, apparently enough to outperform the other two implementations. The waste rate is the ratio of unallocated pixels as a fraction of the total number of pixels in the banner.

As expected there is a strong negative correlation between the waste rate and the profit per banner pixel ($P_{\text{pixel}}$). The obtained value of $-0.9757$ shows that a lower waste rate will result in a higher profit per banner pixel.

In view of the very weak performance of the Greedy stripping 2 and 3 algorithms against the other algorithms, we leave these implementations out of the results in the next sections.

We will now look at the performance in terms of execution time. Table 5.3 presents the distributions of the execution time in seconds for each algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Minimum</th>
<th>Q1</th>
<th>Median</th>
<th>Mean</th>
<th>Q3</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left justified</td>
<td>0.05201</td>
<td>0.18390</td>
<td>1.51700</td>
<td>2.14600</td>
<td>2.99000</td>
<td>19.55000</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>0.04887</td>
<td>0.18770</td>
<td>1.90200</td>
<td>2.21300</td>
<td>3.34800</td>
<td>14.34000</td>
</tr>
<tr>
<td>GRASP con.</td>
<td>0.00606</td>
<td>0.01640</td>
<td>0.06765</td>
<td>0.08175</td>
<td>0.13130</td>
<td>0.31890</td>
</tr>
<tr>
<td>Greedy str. 1</td>
<td>0.00153</td>
<td>0.00345</td>
<td>0.00439</td>
<td>0.00708</td>
<td>0.00973</td>
<td>0.03779</td>
</tr>
<tr>
<td>Greedy str. 2</td>
<td>0.00149</td>
<td>0.00283</td>
<td>0.00460</td>
<td>0.00531</td>
<td>0.00731</td>
<td>0.02930</td>
</tr>
<tr>
<td>Greedy str. 3</td>
<td>0.00149</td>
<td>0.00313</td>
<td>0.00397</td>
<td>0.00673</td>
<td>0.00905</td>
<td>0.03572</td>
</tr>
</tbody>
</table>

It becomes clear that the greedy stripping algorithm is the most efficient of all algorithms. The GRASP constructive algorithm is slower, but still much more efficient than both the left justified and orthogonal algorithm. Notice that the distributions of the latter two algorithms are much wider than the other algorithms. Here, the different approach of the GRASP constructive and greedy stripping algorithm really pays off. The orthogonal and left justified algorithm loose a lot of time by iterating through the banner for every advertisement.

\[^2\text{See Section 4.5 for details.}\]
We can conclude that overall, without specifying a banner size or sorting criteria, the orthogonal algorithm is most effective. In contrast, the greedy stripping algorithm is the most ineffective, but the most efficient.

5.2.2 Sorting

The preliminary sorting of the advertisements influences the final allocation pattern, because all heuristic algorithms iterate through the ordered set of advertisements. In this section we will first focus on the primary sorting criteria, later we will take into account the secondary sorting criteria.

Table 5.4: Profit per pixel for all primary sortings for each algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Profit/pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>10.19</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>10.18</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>9.98</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>9.95</td>
</tr>
<tr>
<td>Left justified</td>
<td>Total area Desc</td>
<td>9.88</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Total area Desc</td>
<td>9.88</td>
</tr>
<tr>
<td>Left justified</td>
<td>Width Desc</td>
<td>9.88</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Total area Desc</td>
<td>9.86</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Width Desc</td>
<td>9.86</td>
</tr>
<tr>
<td>Left justified</td>
<td>Height Desc</td>
<td>9.85</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Height Desc</td>
<td>9.84</td>
</tr>
<tr>
<td>Left justified</td>
<td>Proportionality Asc</td>
<td>9.74</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Proportionality Asc</td>
<td>9.73</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Proportionality Asc</td>
<td>9.67</td>
</tr>
<tr>
<td>Left justified</td>
<td>Flatness Asc</td>
<td>9.66</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Flatness Asc</td>
<td>9.62</td>
</tr>
<tr>
<td>Left justified</td>
<td>Flatness Asc</td>
<td>9.61</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Flatness Desc</td>
<td>9.59</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Width Desc</td>
<td>9.58</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Total area Desc</td>
<td>9.58</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Proportionality Asc</td>
<td>9.56</td>
</tr>
<tr>
<td>Left justified</td>
<td>Flatness Desc</td>
<td>9.56</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Height Desc</td>
<td>9.54</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Flatness Desc</td>
<td>9.52</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Proportionality Desc</td>
<td>9.52</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Flatness Desc</td>
<td>9.49</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Flatness Asc</td>
<td>9.49</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Flatness Asc</td>
<td>9.47</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Proportionality Desc</td>
<td>9.42</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Asc</td>
<td>9.33</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Asc</td>
<td>9.31</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Height Asc</td>
<td>9.31</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Width Asc</td>
<td>9.29</td>
</tr>
<tr>
<td>Left justified</td>
<td>Height Asc</td>
<td>9.26</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Width Asc</td>
<td>9.24</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Asc</td>
<td>9.23</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Total area Asc</td>
<td>9.20</td>
</tr>
<tr>
<td>Left justified</td>
<td>Width Asc</td>
<td>9.19</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Asc</td>
<td>9.16</td>
</tr>
<tr>
<td>Left justified</td>
<td>Total area Asc</td>
<td>9.16</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Total area Asc</td>
<td>9.11</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Width Asc</td>
<td>9.10</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Height Asc</td>
<td>9.05</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Height Asc</td>
<td>9.04</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Total area Asc</td>
<td>8.89</td>
</tr>
</tbody>
</table>
In Table 5.4, the price per pixel is given for all 12 primary sorting criteria, for each algorithm. Note that we still have aggregated the banner sizes and secondary sort. The price per pixel showed is the mean value among all records with the specified algorithm and primary sorting.

From these figures we can conclude that the orthogonal algorithm using a descending price per pixel is the best overall choice with respect to effectiveness, directly followed by the left justified algorithm with the same primary sorting. It also becomes clear that sorting the set of advertisements according to a descending price per pixel, overall yields the best profit per banner pixel irrespective of the algorithm used. After that, sorting by a descending total area is the best for the left justified, orthogonal and GRASP constructive algorithm. Because of this, we can assume that in general it is better to place larger advertisements first, and then filling up the spaces with smaller ones.

Table 5.5 shows the average execution time in seconds per algorithm per primary sorting criteria. In the previous section we already noticed the greedy stripping algorithm outperforming the other algorithms when it comes to execution time. The GRASP constructive algorithm is slower but still far better in comparison with the left justified and orthogonal algorithm.

Furthermore, from the table we can see that most descending sorting criteria outperform the ascending sorting criteria. This goes especially for dimension criteria (i.e., width, height, total area), where placing larger advertisements first yields a lower number of allocated ads, and shortens the execution time. For the greedy stripping and GRASP constructive algorithm on one hand, and the left justified and orthogonal algorithm on the other hand, the ranking per algorithm of the primary sorting criteria in the table is almost the same.

In the previous tables we aggregated the secondary sort. This means the secondary sort was averaged out. However, there may be some combinations of primary and secondary sort that yield better effectiveness than others. We are interested in the influence of the secondary sorting on the results. As an example we show all secondary sorting criteria for all algorithms for a fixed primary sort in Table 5.6. The primary sort is set to a descending price per pixel, since that one was most effective.

Based on these figures, we conclude that the influence on the effectiveness of the secondary sort is relatively small. However, note that $P_{\text{pixel}}$ still has to be multiplied by the total number of pixels to obtain the total profit for a particular banner. In addition we note that the influence of the secondary sort also depends on the set of advertisements and primary sorting criteria used. The secondary sort has more influence when we have a data set with many
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy stripping 1</td>
<td>Total area Desc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Width Desc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Height Desc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Asc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Proportionality Asc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Flatness Desc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Flatness Asc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Proportionality Desc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Height Asc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Width Asc</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Total area Asc</td>
<td>0.01</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Total area Desc</td>
<td>0.05</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Width Desc</td>
<td>0.06</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Height Desc</td>
<td>0.07</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>0.07</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Flatness Desc</td>
<td>0.07</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Asc</td>
<td>0.08</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Proportionality Asc</td>
<td>0.09</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Proportionality Desc</td>
<td>0.10</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Height Asc</td>
<td>0.10</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Flatness Asc</td>
<td>0.10</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Width Asc</td>
<td>0.11</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Total area Asc</td>
<td>0.11</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Height Desc</td>
<td>1.27</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Flatness Asc</td>
<td>1.35</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Height Desc</td>
<td>1.40</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Width Desc</td>
<td>1.40</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Width Desc</td>
<td>1.52</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Flatness Asc</td>
<td>1.69</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Asc</td>
<td>1.71</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>1.78</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Asc</td>
<td>1.79</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>1.81</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Proportionality Asc</td>
<td>1.99</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Proportionality Desc</td>
<td>2.00</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Flatness Desc</td>
<td>2.15</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Total area Desc</td>
<td>2.20</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Proportionality Asc</td>
<td>2.36</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Flatness Desc</td>
<td>2.39</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Total area Desc</td>
<td>2.47</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Proportionality Asc</td>
<td>2.55</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Width Asc</td>
<td>2.93</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Total area Asc</td>
<td>3.14</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Height Asc</td>
<td>3.25</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Height Asc</td>
<td>4.22</td>
</tr>
</tbody>
</table>

Advertisements having the same value for the primary sort, and the secondary sort has to break the ties. Furthermore, these figures still aggregate the banner sizes. For particular banner sizes, other secondary sorts may yield more revenue.

We have seen the performance of the algorithms and sortings with respect to effectiveness (price per pixel) and efficiency (execution time). Now we can propose a combination that yields good performance in general. It is clear that the algorithm should use a descending price per pixel primary sorting. All algorithms proved their effectiveness on this criteria. The orthogonal and left justified algorithm obtained the highest profit per pixel. However, these are
Table 5.6: Influence of secondary sort on Price/pixel Desc primary sort per algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Secondary sort</th>
<th>Price/pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Proportionality Asc</td>
<td>10.2175</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Width Desc</td>
<td>10.2093</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Height Desc</td>
<td>10.2026</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>10.1943</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Flatness Desc</td>
<td>10.1616</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Height Asc</td>
<td>10.1601</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Width Asc</td>
<td>10.1353</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Total area Asc</td>
<td>10.1068</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Width Desc</td>
<td>10.2166</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Proportionality Asc</td>
<td>10.2110</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>10.2017</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Flatness Asc</td>
<td>10.2019</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Height Asc</td>
<td>10.1902</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Proportionality Desc</td>
<td>10.1901</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Total area Asc</td>
<td>10.1851</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Flatness Desc</td>
<td>10.1836</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Height Desc</td>
<td>10.1775</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Width Asc</td>
<td>10.1348</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Proportionality Desc</td>
<td>10.0812</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Flatness Desc</td>
<td>9.9643</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Proportionality Asc</td>
<td>9.9553</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Width Desc</td>
<td>9.9463</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Flatness Asc</td>
<td>9.9422</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>9.9369</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Height Asc</td>
<td>9.9261</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Height Desc</td>
<td>9.9017</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Total area Asc</td>
<td>9.8608</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Width Desc</td>
<td>9.9943</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Flatness Desc</td>
<td>9.9940</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Proportionality Asc</td>
<td>9.9939</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Height Asc</td>
<td>9.9862</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Total area Asc</td>
<td>9.9847</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Proportionality Desc</td>
<td>9.9723</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Width Asc</td>
<td>9.9699</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>9.9659</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Height Desc</td>
<td>9.9629</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Flatness Asc</td>
<td>9.9623</td>
</tr>
</tbody>
</table>

less efficient as the greedy stripping and GRASP constructive algorithm. The efficiency needed is really a personal preference. For our purposes of using the algorithm in a Web application, the orthogonal and left justified algorithm are still far below the specified maximum execution time of 30 seconds. For other uses the need for efficiency might be higher. In that case the greedy stripping and GRASP constructive algorithm are suitable solutions. The differences between both the left justified and orthogonal algorithm are too small to prefer one over the other. For the greedy stripping and GRASP constructive algorithm this is different. If the GRASP constructive algorithm is efficient enough it should be prefered over the greedy stripping algorithm due to its higher effectiveness. However, when this is not the case the greedy stripping algorithm is only choice.
5.2.3 Banner size

In Section 5.2.1 an overview of the overall algorithm’s performance was presented. Note that in the simulation the banner sizes were also reverted to avoid bias towards particular banner sizes. It gives a good indication for the general case, but is not optimal with respect to effectiveness for each banner size. In this section we specify for each banner, for each algorithm, the best setting for the sorting criteria in order to maximize revenue. In addition we give the execution time of the particular algorithms. Recall that we considered the five standard banner sizes “leader board”, “half banner”, “square button”, “skyscraper” and, “large rectangle”. Each size will be discussed in a separate paragraph. In the end, we also discuss the influence of the banner size on the effectiveness.

Leader board 728 x 90

Table 5.7 shows the most effective settings for each algorithm for the leader board. All algorithms are most effective using a descending price per pixel as the primary sort. The left justified and orthogonal algorithm perform equal using a different secondary sort. The GRASP constructive algorithm is most effective using the proportionality as secondary sorting criteria. The greedy stripping algorithm performs the worst with respect to effectiveness.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Secondary sort</th>
<th>Exec. time</th>
<th>( p_{\text{pixel}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Proportionality Asc</td>
<td>1.8888</td>
<td>10.4377</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Height Desc</td>
<td>1.7543</td>
<td>10.4364</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Proportionality Desc</td>
<td>0.1355</td>
<td>10.3576</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>0.0103</td>
<td>10.1450</td>
</tr>
</tbody>
</table>

Half banner 234 x 60

The most effective settings for each algorithm for the half banner are given in Table 5.8. Again, the most effective primary sorting order is the price per pixel descending for all algorithms. The half banner has the same shape as the leader board, resulting in the same ranking of the algorithms. The GRASP constructive and greedy stripping algorithm perform relatively better than on the leader board when it comes to effectiveness.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Secondary sort</th>
<th>Exec. time</th>
<th>( p_{\text{pixel}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Width Desc</td>
<td>0.1092</td>
<td>10.3896</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>0.1488</td>
<td>10.3889</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>0.0136</td>
<td>10.3832</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Proportionality Desc</td>
<td>0.0032</td>
<td>10.2913</td>
</tr>
</tbody>
</table>
Square button 125 x 125

Table 5.9 shows the most effective settings for each algorithm for the square button. For this square shaped banner the orthogonal algorithm is most effective. Furthermore, notice that the most effective left justified algorithm has a descending total area as primary sort. The greedy stripping algorithm prefers the descending width sorting criteria.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Secondary sort</th>
<th>Exec. time</th>
<th>P_{pixel}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Proportionality Desc</td>
<td>0.1875</td>
<td>9.8285</td>
</tr>
<tr>
<td>Left justified</td>
<td>Total area Desc</td>
<td>Price/pixel Desc</td>
<td>0.0748</td>
<td>9.8163</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>0.0094</td>
<td>9.7920</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Width Desc</td>
<td>Price/pixel Desc</td>
<td>0.0023</td>
<td>9.6544</td>
</tr>
</tbody>
</table>

Skyscraper 120 x 600

The most effective settings for each algorithm for the skyscraper are given in Table 5.10. The orthogonal algorithm using a descending width secondary sort is most effective for the skyscraper. For this banner size, the GRASP constructive algorithm has beaten the left justified algorithm. Again, the greedy stripping algorithm is last in the list.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Secondary sort</th>
<th>Exec. time</th>
<th>P_{pixel}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orthogonal</td>
<td>Price/pixel Desc</td>
<td>Width Desc</td>
<td>2.1795</td>
<td>10.6744</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>0.0889</td>
<td>10.6682</td>
</tr>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Total area Desc</td>
<td>1.1622</td>
<td>10.6011</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Price/pixel Desc</td>
<td>Flatness Desc</td>
<td>0.0069</td>
<td>10.4264</td>
</tr>
</tbody>
</table>

Large rectangle 336 x 280

Table 5.11 shows the most effective settings for each algorithm for the large rectangle. For the large rectangle, the left justified algorithm is most effective. This time, the GRASP constructive algorithm beats the orthogonal algorithm. The GRASP constructive algorithm uses a descending total area as primary sort criteria. This is also the only banner for which the orthogonal algorithm is most effective using a different primary sorting order.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Primary sort</th>
<th>Secondary sort</th>
<th>Exec. time</th>
<th>P_{pixel}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left justified</td>
<td>Price/pixel Desc</td>
<td>Proportionality Asc</td>
<td>3.2424</td>
<td>9.7194</td>
</tr>
<tr>
<td>GRASP constructive</td>
<td>Total area Desc</td>
<td>Price/pixel Desc</td>
<td>0.0421</td>
<td>9.7101</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>Total area Desc</td>
<td>Price/pixel Desc</td>
<td>4.0623</td>
<td>9.6722</td>
</tr>
<tr>
<td>Greedy stripping 1</td>
<td>Height Desc</td>
<td>Total area Asc</td>
<td>0.0043</td>
<td>9.4327</td>
</tr>
</tbody>
</table>
Influence of banner size  

From the tables above we can gain some insight in the influence that the banner size has on the profit per banner pixel. Overall, the skyscraper yields the most profit per banner pixel, followed by the leaderboard and half banner. Note that this says nothing about the total revenue the banner creates, since the banners have different sizes. Of course, the placement of the large rectangle banner (total 94080 pixels) creates more total revenue than the other banners.

5.3 Concluding remarks

In this chapter we presented the results of our extensive simulations. First, in Section 5.1 we showed that finding the optimal solution using search algorithms is extremely time-consuming. Using heuristics, comparable results were achieved in short times. Second, we benchmarked the heuristics in Section 5.2. Now we are able to answer the next research subquestions.

What is the most effective heuristic for multiple advertisement allocation?  
We have seen in Section 5.2.1 that the orthogonal algorithm is the choice when not specifying a banner size and sorting criteria. When we do involve sorting criteria like in Section 5.2.2 one would choose the left justified or orthogonal algorithm, using a descending price per pixel primary sort.

What is the most efficient heuristic for multiple advertisement allocation?  
The most efficient heuristic algorithm overall is the greedy stripping algorithm, followed by the GRASP constructive algorithm.

What is the best tradeoff between effectiveness and efficiency for multiple advertisement allocation?  
Choosing a suitable algorithm is done by making a tradeoff between effectiveness and efficiency. There is no best solution since this tradeoff depends on personal preferences. For our purposes, the efficiency of all heuristic algorithms was good enough. Therefore, we focus more on effectiveness than efficiency.

Furthermore, we specified the most effective settings for each algorithm for every banner size used in Section 5.2.3. For the leaderboard (728 × 90) and the large rectangle (336 × 280), the left justified algorithm using a descending price per pixel primary sort and an ascending proportionality secondary sort is most effective. For the half banner (234 × 60) the left justified algorithm using a descending price per pixel primary sort and an descending width secondary sort is most effective. For the square button (125 × 125) the orthogonal algorithm using a descending price per pixel primary sort and an descending proportionality secondary sort is most effective. Finally, for the skyscraper (120 × 600) the orthogonal algorithm using a descending price per pixel primary sort and an descending width secondary sort is most effective.
Chapter 6

Conclusions and Future Research

In this thesis we presented a modified version of the pixel advertisement concept. We focussed on allocating multiple advertisements on a banner. We have gained insight in the different solutions available for this process. In Section 6.1 we will present the conclusions of our research by answering the research questions. In addition, the contributions of this research are set out. Furthermore, suggestions for future research are made in Section 6.2.

6.1 Conclusions

Our main research question was defined as how to efficiently generate the optimal allocation pattern to maximize revenue for multiple advertisement allocation? We have answered the four subquestions from Section 1.1 in the previous chapters. These answers will now be summarized to answer the main research question.

1. How to formally specify the multiple advertisement allocation problem?
   The formal problem definition of the multiple advertisement allocation problem is presented in Section 3.1.

2. What is the most effective heuristic for multiple advertisement allocation?
   The orthogonal algorithm is the choice without specifying a banner size and sorting criteria, as we have seen in Section 5.2.1. If we take the primary sorting criteria into account, the most effective heuristics are the left justified and orthogonal algorithm using a descending price per pixel primary sort. Furthermore, the most effective settings for each banner
3. What is the most efficient heuristic for multiple advertisement allocation? The most efficient heuristic algorithm is the greedy stripping algorithm, followed by the GRASP constructive algorithm. Details can be found in Section 5.2.

4. What is the best tradeoff between effectiveness and efficiency for multiple advertisement allocation? Choosing the best heuristic comes down to making a tradeoff between effectiveness and efficiency. One should find a good balance that fits personal preferences. For our goals, the efficiency of all heuristics was good enough, so we focus on effectiveness. Then the orthogonal or left justified algorithm, using a descending price per pixel as primary sorting criteria, is the best choice. A detailed description of the results is available in Section 5.2.

The problem of multiple advertisement allocation has not received much attention in literature. Therefore, this paper has a large contribution to this field, especially on the following points.

- Several heuristic algorithms are proposed that provide adequate solutions for the problem.
- A benchmark of heuristic algorithms is added, specifying the best algorithms and settings for particular banner sizes.
- An analysis of the effectiveness-efficiency tradeoff in multiple advertisement allocation is added.

6.2 Future Research

This research also uncovers possible future work directions. These research directions are identified below.

Our research is limited to the allocation algorithms we have used. Other heuristics may yield a better effectiveness-efficiency tradeoff. The left justified algorithm in Section 4.2 and the orthogonal algorithm in Section 4.3 could also be improved. These algorithms are quite straightforward in that they iterate through the banner every time. They are revisiting places that are known to be occupied. Saving this information and thereby preventing revisits could make the algorithms even more efficient.

It might be more realistic to give different positions on a banner different prices. In our research we have a predefined set of advertisements with different prices regardless of the position they get allocated. The Eyetrack III [18]
research investigates people's eye movements over Web pages. More frequently watched areas in the banner may be assigned a higher price.

Furthermore, the multiple advertisement allocation problem can be extended to a scheduling problem. Until now, related work only focused on scheduling advertisements side-by-side, instead of banners that allocate in a two-dimensional way. Scheduling makes the banner content dynamic by adding time slots and will increase user attention.
Bibliography


