Maximizing revenue with allocation of multiple advertisements on a Web banner

Victor Boskamp, Alex Knoops, Flavius Frasincar[*], Adriana Gabor

Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, the Netherlands

Abstract

The problem addressed in this paper is the allocation of multiple advertisements on a Web banner, in order to maximize the revenue of the allocated advertisements. It is essentially a two-dimensional, single, orthogonal, knapsack problem, applied to pixel advertisement. As this problem is known to be NP-hard, and due to the temporal constraints that Web applications need to fulfill, we propose several heuristic algorithms for generating allocation patterns. The heuristic algorithms presented in this paper are the left justified algorithm, the orthogonal algorithm, the GRASP constructive algorithm, and the greedy stripping algorithm. We set out an experimental design using standard banner sizes, and primary and secondary sorting criteria for the set of advertisements. We run two simulations, the first simulation compares the heuristics with an optimal solution found using brute force search, and the second simulation compares the heuristic algorithms to gain a better insight into their performance. Finding a suitable pattern generating algorithm is a tradeoff between effectiveness and efficiency. Results indicate that allocating advertisements with the orthogonal algorithm is the most effective. In contrast, allocating advertisements using the greedy stripping algorithm is the most efficient. Furthermore, the best settings per algorithm for each banner size are given.

Key words: Pixel advertisement, Allocation patterns, Heuristics, Two-dimensional knapsack problem

1. Introduction

With the continuing growth of Web usage, Web advertising becomes a more dominant form of marketing every year. According to the Interactive Advertising Bureau, Web advertising revenues for 2008 totaled $23.4 billion in the U.S. only [1]. A special form of Web advertising is *pixel advertisement*, 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 displays 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 advertisement image on their pixels and let it link to their website. The pixel advertisement website became a great success, due to its novel proposition [3]. The copycats that arose could not repeat the success of the "Million Dollar Homepage", but applying the concept in a different way may still be interesting. In this work, we apply the pixel advertisement 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.

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, while here not every advertisement is placed on the banner. Each advertisement has a certain price per pixel, which can be obtained either by negotiations with the space-owner or just represent the value the advertiser is asked to pay for it. If an advertisement is placed, the advertiser has to pay the costs corresponding to the size of the advertisement. An assumption we make is that we have more advertisements

[*]Corresponding author. Tel. +31 10 40 81 340; fax: +31 10 40 89 162

Email addresses: `victorboskamp@gmail.com` (Victor Boskamp), `alex.knoops@gmail.com` (Alex Knoops), `frasincar@ese.eur.nl` (Flavius Frasincar), `gabor@ese.eur.nl` (Adriana Gabor)

Figure 1: Sample from "Million Dollar Homepage"

than would fit on the banner, so some advertisements of the set may not be placed. This increases the competition among advertisers which will lead to higher prices per pixel. On the other hand, smaller advertisements are more affordable, which makes Internet advertising more accessible for anyone. When we have a set of advertisements, we allocate them across the banner in such a way that the revenue of the space-owner is maximized. We name this modified version of the pixel advertisement concept, *multiple advertisement allocation*.

This research deals specifically with the advertisement allocation process and focuses on finding a suitable pattern generating algorithm. The problem tackled in this work can be seen as a variant of the well-known cutting and packing problems in the literature. When following the typology in [4], our problem can be classified as a *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 *single* means we have only one large object to place our advertisements in, namely the banner. In *orthogonal* patterns, the edges of small items are set parallel to the edges of the large object and rotation is not allowed [5]. The problem is NP-hard [6], making it extremely time-consuming to find the optimal solution. Besides this, we also plan to apply our solution on the Web which makes temporal restrictions highly relevant. As a consequence of all these constraints, we decided to apply heuristics to find adequate solutions.

Due to the nature of our problem we chose a simulation-based research methodology. We implement several heuristic algorithms for generating adequate allocation patterns for multiple advertisement placement. In addition, we implement a brute force search algorithm that finds the optimal allocation pattern. In our experiments we run two simulations, a small and a large one. The small simulation benchmarks the heuristics against the brute force search algorithm using a small problem size, in order to emphasize the difficulties of using an optimal solution yielding approach under temporal constraints. The large simulation uses a normal problem size with only heuristic algorithms, to gain a better insight into their performance. We analyse the performance of the different algorithms with respect to their effectiveness and efficiency.

In this paper, we extend our previous work on pixel advertisement [7] by refining the experimental design. We use more realistic sets of advertisements, improved the GRASP algorithm, and added two other algorithms (i.e., the brute force search algorithm, and the greedy stripping algorithm). In addition, we do a more thorough analysis of the effectiveness and efficiency of the algorithms, also covering the influence sorting criteria and banner types have on the results.

The rest of this paper is organized as follows. First, we discuss related work in Section 2. Then, we give the problem formulation and we present a brute force search implementation for finding the optimal solution in Section 3. Four heuristic algorithms that generate adequate solutions are presented in Section 4. The description of the experimental design is given in Section 5. In Section 6 we present and analyse the results of the simulation experiments. Finally, Section 7 concludes the paper and identifies future work directions.

2. Related work

At the current moment, there is very little literature available on pixel advertisement. In [3], Web advertisement is discussed in general and the "Million Dollar Homepage" is analysed. The author identifies the success factors and also proposes some improvements for the pixel advertisement concept. Such an improvement is a heuristic approach for placing multiple advertisements on a banner in [8]. We formalized this heuristic-based approach and performed an extensive evaluation in [7].

Other literature on the placement of Web advertisements has been focusing on the *ad placement problem*, introduced in [9] 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 it is cycled over time, and every slot has a different allocation pattern of advertisements. Furthermore, the authors are concerned only with the placement of one advertisement on a banner or some advertisements side-by-side, with the height of the advertisements equal to the height of the banner. Note that this differs from the pixel advertisement and the multiple advertisement allocation approach, where advertisements are placed in a two-dimensional way.

Since the APP is the most popular related problem we give a short overview of the field aiming to solve this problem. In [9], 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, whereas in the online problem we receive requests for placement sequentially and have to decide whether to accept these without knowledge of future requests. Another distinction made concerns the *MINSPACE* and *MAXSPACE* problem. In the MINSPACE problem, we are given a number of time slots and a set of advertisements and have to allocate all advertisements over the time slots, while minimizing the banner size. Note that all time slots have the same width and height. The goal of the MINSPACE problem is to find the minimum banner width and height such that all advertisements are placed. In the MAXSPACE problem, we are given the banner dimensions, the number of time slots, and a set of advertisements, and we have to allocate the advertisements in such a way that the revenue is maximized, which means that not all advertisements are allocated. The goal of the MAXSPACE problem is to find the optimal allocation pattern. For both problems, several solutions are available using polynomial time approximation algorithms [10, 11, 12], column generation [13], Lagrangian decomposition [13, 14], and a hybrid genetic algorithm [15]. Based on the previous classifications, our research deals with an offline and MAXSPACE problem.

An overview and categorisation of cutting and packing problems is given in [4]. In this categorisation, several variants of the knapsack problem are identified, which differ in the dimension used, the size and the number of containers, or the use of rotated items. According to this classification, our problem is a two-dimensional, single, orthogonal, knapsack problem. As previously stated in the introduction, the problem is two-dimensional as the banner and advertisements have only two dimensions (width and height). It is a single problem as we have only one container, the banner. The problem is orthogonal as we are allowed to place advertisement edges only parallel to the banner edges. Last, it is a knapsack problem as we have more advertisements to place than the banner supports.

There are several exact approaches available on the *two-dimensional knapsack problem* (2DKP) and related cutting and packing problems. The solution space of these problems can be seen as a tree, where enumerative algorithms are used to find the optimal solution. In [5] and [16], exact tree search procedures are presented for the *two-dimensional orthogonal knapsack problem*. The algorithms limit the search space by using an upper bound on the optimal solution, after which branch & bound is used to solve the problem. A related problem, the *two-staged two-dimensional knapsack problem*, is considered in [17]. The two stages refer to the classical variant of the 2DKP, in which the maximum number of cuts allowed to obtain each item is fixed to 2. The authors propose two integer linear programming models that are solved with a branch & bound algorithm, one for an unconstrained version of the problem, that has no limit to the number of items that can be cut off, and another in which a 90° rotation of items is allowed.

Another problem related to the 2DKP is the *two-dimensional orthogonal packing problem* (2D-OPP). This problem, classified as an open dimension problem, differs from the knapsack problem in the sense that it focuses on input minimisation, whereas the knapsack problem focuses on output maximization [4]. In [18], the authors present two exact algorithms for this problem, the first being an improvement on the classical branch & bound approach, the second is based on a new relaxation of the problem. An exact branch & bound algorithm for the subproblem of finding a set of items that fits into the container, the *orthogonal packing problem*, is presented in [19]. The authors assign items to the container without specifying a position, and solve a packing problem after each assignment using a graph representation to find the optimal packing for the container. A branch & bound tree search procedure is also at the basis of the study in [20], that considers a *non-guillotine cutting problem* (NGCP). Cutting problems focus on input minimisation, their objective is to allocate all items to a selection of large objects such that the value, number, or total size of these objects is minimized [4]. Non-guillotine cutting means that cutting is not performed from edge-to-edge of the object, like is done in guillotine cuts [21].

Other solutions are based on heuristics for solving the cutting and packing problems. The *two-dimensional single knapsack problem* is solved in [22] by using a local search procedure controlled by simulated annealing. In [23], the authors present an efficient heuristic for solving the 2DKP. Their approach consists of solving a sequence of one-dimensional knapsack problems, such that the obtained optimal subsets create a near-optimal solution for the two-dimensional knapsack problem. The authors also derive an approximation algorithm from the heuristic. In [24], the authors present heuristic algorithms for the *multiple-choice multidimensional knapsack problem*. In this variant of the knapsack problem, there are several classes of items, with the goal to pick one item from each class such that the revenue is maximized. Their approach consists of a constructive heuristic that builds an initial solution, and a local search improvement heuristic that tries to improve the solution by replacing and swapping items.

For the 2D-OPP, a hybrid metaheuristic approach is presented in [25], combining a genetic algorithm to decide on the packing order, and a constructive placement heuristic. The two-dimensional packing problem is used to refer to both the *two-dimensional bin packing problem* (2D-BPP) and the *two-dimensional strip packing problem* (2D-SPP). In [26], the authors present heuristic algorithms for four cases of the 2D-BPP, and a unified tabu search approach. An extensive survey on both the 2D-BPP and 2D-SPP is available in [27]. Heuristic approaches for the NGCP are presented in [28] and [29], that use evolutionary algorithms, and in [30] using simulated annealing. Furthermore, in [31] the authors use a *greedy randomized adaptive search procedure* (GRASP) to find the solution to the problem. GRASP algorithms consist of a greedy randomized constructive phase, and a local search-based improvement phase. The same authors also present a tabu search algorithm for solving the problem in [32].

3. Problem definition and optimal solution

The purpose of this section is to specify the problem that we try to solve, and to discuss the process of finding the optimal solution. First, we give the formal problem definition together with an integer programming formulation in Section 3.1. Second, we elaborate upon the optimal solution and present a brute force search algorithm for finding the optimal solution in Section 3.2.

3.1. Problem definition

The formal definition of the problem to be solved is as follows. We have a set $\mathcal{A}$ with a fixed number of advertisements $|\mathcal{A}|$ to allocate in a banner. We assume we have more ads in $\mathcal{A}$ than would fit on the banner, thus not every advertisement is placed. Each advertisement $a_i \in \mathcal{A}$ has a width w_i, height h_i, and a price per pixel pp_i, with $i \in \{1, \ldots, |\mathcal{A}|\}$. The banner has width $\mathcal{W}$ and height $\mathcal{H}$. The advertisements from $\mathcal{A}$ should be allocated on the banner such that the total value of the set of allocated advertisements $\mathcal{A}'$ (subset of $\mathcal{A}$) is maximized. Each advertisement a_i in $\mathcal{A}'$ has its top-left corner at position (x, y) on the banner, starting from $(0, 0)$ which represents the top-left corner of the banner. The value of an allocated advertisement in $\mathcal{A}'$ is defined by v_i, where $v_i = pp_i \times w_i \times h_i$. Our objective is to maximize the total value of allocated advertisements in $\mathcal{A}'$.

We can formulate the problem as a 0-1 integer programming problem, which is a simplification of the problem formulation from [5]. The possibility of having replicates has been removed from the original formulation in [5], since our problem assumes every advertisement can be allocated only once. In order to make sure that advertisements do not overlap on the banner we have reused a constraint from [20]. Let

$$\mathcal{X}_i = \{x \mid 0 \leq x \leq \mathcal{W} - w_i\}, \ \forall\, i \in \{1, \ldots, |\mathcal{A}|\}$$

be the set of all possible points x along the width of the banner such that an advertisement a_i from $\mathcal{A}$ can be placed on the banner with its top-left corner at x-position x, $x \in \mathcal{X}_i$. Similarly we define

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

as the set of all possible allocation points y along the height of the banner. We define

$$x_{ip} = \begin{cases} 1 & \text{if } a_i \text{ is assigned to the banner with its top-left corner at x-position } p \text{ where } p \in \mathcal{X}_i \\ 0 & \text{otherwise} \end{cases}$$

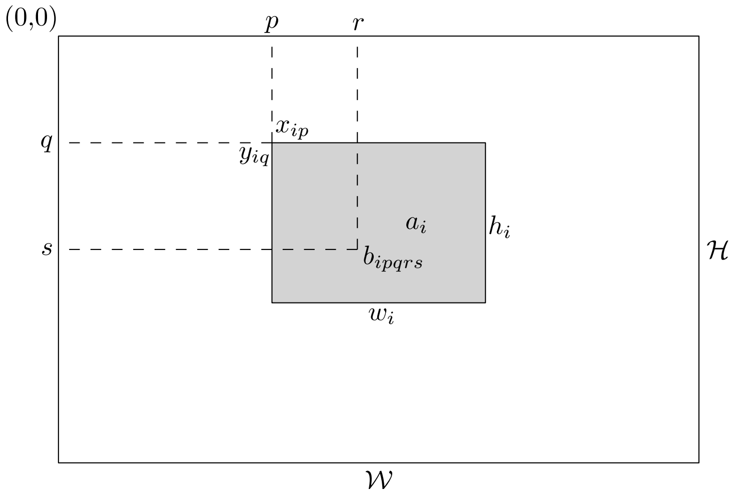

Figure 2: Visualization of x_{ip}, y_{iq}, and b_{ipqrs}

$$y_{iq} = \begin{cases} 1 & \text{if } a_i \text{ is assigned to the banner with its top-left corner at y-position } q \text{ where } q \in \mathcal{Y}_i \\ 0 & \text{otherwise} \end{cases}$$

and let

$$b_{ipqrs} = \begin{cases} 1 & \text{if advertisement } i\text{, when placed with its top-left corner at position } (p,q)\text{, cuts out point } (r,s) \text{ of the banner} \\ 0 & \text{otherwise} \end{cases}$$

which can be restated as

$$b_{ipqrs} = \begin{cases} 1 & \text{if } 0 \le p \le r \le p + w_i - 1 \le \mathcal{W} - 1 \text{ and } 0 \le q \le s \le q + h_i - 1 \le \mathcal{H} - 1 \\ 0 & \text{otherwise} \end{cases}$$

Figure 2 visualizes x_{ip}, y_{ip}, and b_{ipqrs} with respect to the banner. Now, the integer programming formulation can be stated as follows:

$$\max \sum_{i=1}^{|\mathcal{A}|} v_i \sum_{p \in \mathcal{X}_i} x_{ip} \tag{1}$$

subject to

$$\sum_{i=1}^{|\mathcal{A}|} \sum_{p \in \mathcal{X}_i} \sum_{q \in \mathcal{Y}_i} b_{ipqrs} x_{ip} y_{iq} \le 1, \ \forall\, r \in \{0, \ldots \mathcal{W} - 1\}, \ \forall\, s \in \{0, \ldots \mathcal{H} - 1\} \tag{2}$$

$$\sum_{p \in \mathcal{X}_i} x_{ip} \le 1, \ \forall i \in \{1, \ldots, |\mathcal{A}|\} \tag{3}$$

$$\sum_{p \in \mathcal{X}_i} x_{ip} = \sum_{q \in \mathcal{Y}_i} y_{iq}, \ \forall i \in \{1, \ldots, |\mathcal{A}|\} \tag{4}$$

$$x_{ip}, y_{iq} \in \{0, 1\}, \ \forall\, i \in \{1, \ldots, |\mathcal{A}|\}, \ \forall\, p \in \mathcal{X}_i, \ \forall\, q \in \mathcal{Y}_i \tag{5}$$

In Eq. (1) the objective function maximizes the total value of the allocated advertisements. Constraint (2) ensures that any banner point is used by at most one advertisement. Constraints (3) and (4) ensure that any advertisement is allocated at most once on the whole banner. The ranges of p and q, i.e., $\mathcal{X}_i$ and $\mathcal{Y}_i$, respectively, ensure that advertisements are always placed inside the banner. (5) is the integrality constraint. The model can be linearized by replacing variables x_{ip} and y_{iq} with a variable z_{ipq} as shown in [20].

3.2. 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 that can be formulated as an integer program.

To obtain the optimal solution, search algorithms may be used. These algorithms investigate the search space, i.e., the set of all feasible solutions to a problem, and look for the best one. Search algorithms can be either uninformed or informed. Uninformed search algorithms try all possible solutions in the search space, while informed search algorithms use heuristics to apply knowledge about the structure of the search space to reduce the execution time.

We use a standard uninformed brute force search to obtain the optimal solution. 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 [34]. The algorithm is used in many problems, all with a slightly different implementation to fit the problem specifics. Our algorithm is presented in Algorithm 1. The algorithm is recursive; on every location of the banner we try to place every advertisement once. The recursion makes sure we generate all possible allocation patterns. Every new optimal allocation pattern is stored, eventually yielding one or more solutions which maximize profit.

Finding the optimal solution by exhaustive search for a realistic problem size, like we present in Section 5, is extremely time consuming. With possible Web use in mind, we want the maximum execution time of the allocation process to be under 30 seconds. Therefore we propose in Section 4 a few heuristic algorithms that compute solutions close to the optimal one in a relatively small amount of time. To give an indication of the differences in effectiveness and efficiency between the exhaustive search algorithm and heuristic-based search algorithms, we compare their performance on a problem of decreased size (for computational reasons), which is presented together with the comparison results in Section 6.

4. Four heuristic-based algorithms

In this section we present four heuristic-based algorithms for the multiple advertisement allocation problem. First we give the initialization for our heuristic algorithms. Then, we present the *left justified algorithm*, the *orthogonal algorithm*, the *GRASP constructive algorithm*, and the *greedy stripping algorithm*.

4.1. Initialization

The initialization step considers the sorting of the set of advertisements $\mathcal{A}$, and is identical for all heuristic algorithms. We use a sorted list of advertisements, since the order in which the advertisements are processed influences the pattern generated by the heuristic algorithms. Apart from a random order, we propose several sorting criteria based on advertisement specifics, like price per pixel and total area. We sort the set of advertisements according to a primary and secondary sort, s_1 and s_2. Their values may be either positive or negative depending on an ascending or descending sorting order. The primary and secondary sort may not apply the same criteria, and we also avoid duplicate sorting in the opposite direction. Sorting $\mathcal{A}$ according to s_1 and s_2 yields $\mathcal{A}_0$, the list of advertisements through which is iterated with variable i.

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. Parameters q and p are cursor-variables, corresponding to the row and column position in the banner, respectively, when the banner is seen as a matrix. The variable w_i represents the width of an advertisement, h_i is the height of the advertisement, and $|\mathcal{A}|$ is the number of advertisements in the set $\mathcal{A}$.

```
function brute-force-search($\mathcal{A}$, banner, $\mathcal{W}$, $\mathcal{H}$, $q$, $p$, $maxprofit$)
{
while $p < \mathcal{W}$ do
    while $q < \mathcal{H}$ do
        if $banner_{q,p} = 0$ then
            {Location is free}
            $i := 1$;
            while $i \leq |\mathcal{A}|$ do
                Get $a_i$ from $\mathcal{A}$;
                if $a_i$ has not been allocated already then
                    if $a_i$ fits in $banner_{q,p}$ then
                        Allocate $a_i$ in $banner_{q,p}$;
                        Mark $a_i$ in $\mathcal{A}$ as allocated;
                        Compute total $profit$ of $banner$;
                        if $profit \geq maxprofit$ then
                            $maxprofit := profit$;
                            Store allocation pattern;
                        end if

                        brute-force-search($\mathcal{A}$, banner, $\mathcal{W}$, $\mathcal{H}$, $q$, $p$, $maxprofit$);

                        Remove advertisement $a_i$ from banner;
                        Unmark $a_i$ in $\mathcal{A}$ as allocated;
                    end if
                end if
                $i := i + 1$;
            end while
        end if
        $q := q + 1$;
    end while
    $p := p + 1$;
    $q := 0$;
end while
}
```

4.2. *Left justified algorithm*

The *left justified algorithm* iterates through the list of advertisements $\mathcal{A}_0$. For each advertisement a_i it scans through the columns of the banner 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. When an available field is found and a_i fits on the empty location, the advertisement is placed in the banner 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 list $\mathcal{A}_0$, the algorithm stops and the allocation pattern is returned. The details of this algorithm are shown in Algorithm 2.

4.3. *Orthogonal algorithm*

The *orthogonal algorithm* iterates through the list of advertisements $\mathcal{A}_0$ and places advertisements as close as possible to the top left corner. The algorithm looks for free locations for the current advertisement by moving diagonally from the top left corner $(q, p) = (0, 0)$ of the banner. At each step, the algorithm searches at location (q, p') with $p' \in \{0 \dots p\}$ and (q', p) with $q' \in \{0 \dots q\}$ for the first free space where the advertisement can be allocated. We store the first free location from the borders to (q, p) in variables *verticalplace* and *horizontalplace*. After that we compare these variables with respect to the sum of the distances to the top and to the left and allocate a_i on the position that

Algorithm 2 *Left justified algorithm*

```
Run heuristic algorithm initialization;
for $i := 1$ to $|\mathcal{A}|$ do
    Get $a_i$ from $\mathcal{A}_0$;
    $finished := false$;
    $q := 0$; {Current row in $banner$}
    $p := 0$; {Current column in $banner$}
    while $finished = false$ do
        if $a_i$ fits in $banner_{q,p}$ then
            Allocate $a_i$ in $banner_{q,p}$;
            $finished := true$;
        else if $q + h_i > \mathcal{H}$ then
            {$a_i$ goes vertically out of bounds}
            if $p < \mathcal{W} - 1$ then
                $p := p + 1$;
                $q := 0$;
            else
                $finished := true$;
            end if
        else if $p + w_i > \mathcal{W}$ then
            {$a_i$ goes horizontally out of bounds}
            $finished := true$;
        else
            {Pixels needed for placement $a_i$ are already occupied}
            if $q < \mathcal{H} - 1$ then
                $q := q + 1$;
            else
                if $p < \mathcal{W} - 1$ then
                    $p := p + 1$;
                    $q := 0$;
                else
                    $finished := true$;
                end if
            end if
        end if
    end while
end for
return $banner$;
```

yields the smallest sum. When there is a tie we choose the one on the vertical search path. When a_i is allocated, we start again in the top-left corner of the banner, trying to allocate a_{i+1}.

When we fail to allocate an advertisement for a certain location (q, p) we continue to walk diagonally down-right by increasing both q and p variables by one. When the final row is reached, but there are still columns left, we only increase column cursor p. When the final column is reached, but there are still rows left, we only increase row cursor q. This means that after we start walking diagonally, we will eventually switch to walking either right or down, except for the situation in which the banner is a square.

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 $\mathcal{A}_0$. When we iterated through all advertisements in $\mathcal{A}_0$ the algorithm stops. The details of this algorithm are shown in Algorithm 3.

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 [31]. In GRASP, an iterative procedure combines a constructive phase and an improvement phase. In the constructive phase, a solution is built 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 algorithms discussed previously, the only thing they have in common is the initialization based on the sorting of advertisements. Sorting the advertisements to certain criteria makes the algorithm

Algorithm 3 *Orthogonal algorithm*

```
Run heuristic algorithm initialization;
for $i := 1$ to $|\mathcal{A}|$ do
    Get $a_i$ from $\mathcal{A}_0$;
    $finished := false$; {To stop searching locations for this $a_i$}
    $q := 0$; {Current row in $banner$}
    $p := 0$; {Current column in $banner$}
    $rowscompleted := false$; $colscompleted := false$;
    $verticalfound := false$; $horizontalfound := false$;
    $verticalplace := (0, 0)$; $horizontalplace := (0, 0)$; {To store candidate locations}
    $rowposition := 0$; $colposition := 0$; {To store chosen location}
    while $finished = false$ or ($rowscompleted$ and $colscompleted$) $= false$ do
        if $colscomplete = false$ then
            {Search column from top border to cursor for candidate location}
            for $q' := 0$ to $q$ do
                if $a_i$ fits in $banner_{q',p}$ then
                    $verticalplace := (q', p)$; $verticalfound := true$; Break out of for-loop;
                end if
            end for
        end if
        if $rowscomplete = false$ then
            {Search row from left border to cursor for candidate location}
            for $p' := 0$ to $p$ do
                if $a_i$ fits in $banner_{q,p'}$ then
                    $horizontalplace := (q, p')$; $horizontalfound := true$; Break out of for-loop;
                end if
            end for
        end if
        if $horizontalfound = true$ or $verticalfound = true$ then
            if $horizontalfound = true$ and $verticalfound = true$ then
                {Select location closest to left or top border}
                if $q' + p$ ($verticalplace$) $\leq q + p'$ ($horizontalplace$) then
                    $rowposition := q'$; $colposition := p$;
                else
                    $rowposition := q$; $colposition := p'$;
                end if
            else if $verticalfound = true$ then
                $rowposition := q'$; $colposition := p$;
            else if $horizontalfound = true$ then
                $rowposition := q$; $colposition := p'$;
            end if
            Allocate $a_i$ in $banner_{rowposition,colposition}$;
            $finished := true$;
        else
            if $q < \mathcal{H} - 1$ then
                $q := q + 1$;
            else
                $rowscompleted := true$;
            end if
            if $p < \mathcal{W} - 1$ then
                $p := p + 1$;
            else
                $colscompleted := true$;
            end if
        end if
    end while
end for
return $banner$;
```

greedy. The main difference with the other algorithms is that we don't search the banner for free space, but store rectangles of free space in set $\mathcal{L}$. Free rectangles are parts of the banner where no advertisement is allocated yet. Initially, set $\mathcal{L}$ contains only the full banner.

Algorithm 4 *GRASP constructive algorithm*

```
Run heuristic algorithm initialization;
$finished := false$; {Boolean variable to stop the algorithm}
while $finished = false$ do
    $\mathcal{L}_j :=$ smallest rectangle from $\mathcal{L}$ not marked as used;
    $ad\_placed := false$;
    $i := 1$;
    while ($ad\_placed = false$) and ($i \leq |\mathcal{A}_0|$) do
        Get $a_i$ from $\mathcal{A}_0$;
        if $a_i$ fits in $\mathcal{L}_j$ then
            Allocate $a_i$ in the banner in $\mathcal{L}_j$;
            Remove $a_i$ from $\mathcal{A}_0$;
            $ad\_placed := true$;
            if $\mathcal{L}_j$ is not completely filled by $a_i$ then
                Cut new rectangles;
                Add new rectangles to $\mathcal{L}$;
            end if
            Remove rectangle $\mathcal{L}_j$ from $\mathcal{L}$;
        else
            $i := i + 1$;
        end if
    end while
    if $ad\_placed = true$ then
        $nrectangles :=$ number of rectangles in $\mathcal{L}$;
        if $nrectangles \geq 2$ then
            $rectangleA := 1$;
            while $rectangleA \leq nrectangles - 1$ do
                $rectangleB := rectangleA + 1$;
                while $rectangleB \leq nrectangles$ do
                    if rectangleA and rectangleB are adjacent then
                        Merge rectangleA and rectangleB;
                        Add new rectangle to $\mathcal{L}$;
                        Remove rectangleA and rectangleB from $\mathcal{L}$;
                        $nrectangles := nrectangles - 1$;
                        $rectangleA := 1$;
                        Break out of while-loop;
                    end if
                    $rectangleB := rectangleA + 1$;
                end while
                $rectangleA := rectangleA + 1$;
            end while
        end if
    end if
    Mark $\mathcal{L}_j$ as used;
    if all rectangles in $\mathcal{L}$ are marked as used then
        $finished := true$;
    end if
end while
return banner;
```

To allocate advertisements, the following procedure is followed. First, we take the smallest rectangle of $\mathcal{L}$ in which an advertisement from list $\mathcal{A}_0$ can fit. Then, we place an advertisement a_i from list $\mathcal{A}_0$ that fits in the free rectangle. Whenever an advertisement is placed in a rectangle, new free rectangles are formed and added to $\mathcal{L}$, while the original rectangle is removed from $\mathcal{L}$. 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 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 first advertisement from list $\mathcal{A}_0$. 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 $\mathcal{A}_0$ in a rectangle from $\mathcal{L}$, we mark the rectangle as *used*. This allows us to eventually stop the algorithm.

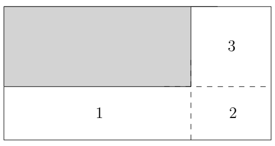

Figure 3: Free rectangles in GRASP algorithm

The GRASP constructive algorithm differs from the original GRASP approach, since we integrate the constructive phase and improvement phase to decrease the execution time. To enlarge the chance for free rectangles to be allocated by an advertisement, we directly merge adjacent free rectangles after they appear in $\mathcal{L}$. The new free rectangle is then added to $\mathcal{L}$ and the merged rectangles are removed from $\mathcal{L}$. 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 to improve it in the improvement phase using a local search procedure. An example of this procedure is to start by removing a number of advertisements from the solution, merging and shifting free rectangles, and finally adding new advertisements to the solution. This process is repeated until the solution does not improve any more. For our application, merging empty rectangles "on the go" is more efficient than introducing a local search procedure to improve the solution.

After we have processed a rectangle we continue with the next smallest rectangle from $\mathcal{L}$ not used before. When there are no free rectangles left ($\mathcal{L}$ is empty, the full banner is allocated) or no advertisements from list $\mathcal{A}_0$ that fit any of the remaining rectangles, the algorithm stops. A brief overview of this algorithm is shown in Algorithm 4.

4.5. Greedy stripping algorithm

In the *greedy stripping algorithm*, advertisements are allocated in *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 and if the banner is flat we create vertical strips. We start by placing the first advertisement a_i from $\mathcal{A}_0$, and its width w_i or height h_i, 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 $\mathcal{A}_{sub}$, which contains $|\mathcal{A}_{sub}|$ advertisements. The subset is then ordered according to width or height in descending order, depending on the shape of the banner, flat or tall, respectively. After that, we iterate through $\mathcal{A}_{sub}$ trying to place the advertisements in the strip. Whenever an advertisement a_i from $\mathcal{A}_{sub}$ is allocated, we remove it from original list $\mathcal{A}_0$. If we reached the end of the strip or $\mathcal{A}_{sub}$ is empty we create a new strip. Since $\mathcal{A}_0$ has changed after filling a strip, we start the next strip with the first advertisement from $\mathcal{A}_0$. We continue doing this until we iterated through all advertisements from $\mathcal{A}_0$, then the algorithm stops. The details of this algorithm are shown in Algorithm 5, where we present the case for which the banner is tall. When the banner is flat or a square, an analogue procedure can be followed exchanging the row and height-variables with the col and width-variables.

5. Experimental design

We run two simulations in which algorithms for multiple advertisement allocation are compared. First, we show that finding the optimal solution is extremely time-consuming by benchmarking the heuristic algorithms against the brute force search algorithm. Second, we run a large simulation for comparing the heuristic algorithms.

In the heuristics simulation, every simulation cycle has a different configuration of its parameters. We distinguish the following three configuration parameters of which one is changed in every simulation cycle.

- Size of the banner;
- Sorting of the advertisements;
- Algorithm.

All possible combinations of values from these parameters are considered in the simulation in order to obtain unbiased results. The order in which the configuration parameters are given specifies the setup of the simulation.

Algorithm 5 *Greedy stripping algorithm*

```
Run heuristic algorithm initialization;
$i := 1$;
$q := 0$; {Current row in *banner*}
$p := 0$; {Current column in *banner*}
if $\mathcal{W} < \mathcal{H}$ then
    {*banner* is a *tall* banner, we create horizontal strips}
    $next\_strip\_top\_location := 0$; {Keeps track of next strip location}
    while $\mathcal{A}_0 \neq \emptyset$ do
        Get $a_i$ from $\mathcal{A}_0$;
        $found\_ad := false$;
        $q := next\_strip\_top\_location$;
        if $next\_strip\_top\_location + h_i > \mathcal{H}$ then
            if $i \geq |\mathcal{A}|$ then
                Break out of while-loop; {All ads have been checked, stop algorithm}
            end if
            $i := i + 1$; {Try next advertisement}
        else
            $found\_ad := true$; {We start a new strip with this $a_i$}
            $next\_strip\_top\_location := next\_strip\_top\_location + h_i$;
        end if
        if $found\_ad = true$ then
            Create subset $\mathcal{A}_{sub}$ of $\mathcal{A}_0$ with advertisements that have same or lower height $h_i$;
            Sort $\mathcal{A}_{sub}$ according to descending $h_i$;
            for $k := 1$ to $|\mathcal{A}_{sub}|$ do
                Get $a_k$ from $\mathcal{A}_{sub}$;
                if $a_k$ fits in $banner_{q,p}$ then
                    Allocate $a_k$ in $banner_{q,p}$;
                    Remove $a_k$ from $\mathcal{A}_0$;
                    if $p + w_k < \mathcal{W}$ then
                        $p := p + w_k$; {Next free location}
                    else
                        $p := 0$; {Start with new strip}
                        $i := 1$; {$\mathcal{A}_0$ has changed, start on top}
                        Break out of for-loop;
                    end if
                    $i := 1$; {$\mathcal{A}_0$ has changed, start on top}
                end if
            end for
            $p := 0$; {We have iterated through all ads in $\mathcal{A}_{sub}$, start with new strip}
        end if
    end while
else
    Do the same as above for the *flat* or *square* banner, but this time with vertical strips instead of horizontal strips;
    {This part of the code is analogue to the part above, except now all width-variables are exchanged with height-variables}
end if
return *banner*;
```

For every banner size we simulate all considered sorts of the set of advertisements, and for every sort we run all algorithms. During each cycle of the simulation we register the configuration parameters, the execution time, the number of advertisements placed, and calculate the total profit of the generated allocation pattern. The experiment is implemented in Matlab R2008b and run on an Intel Core 2 Quad CPU at 2.40 GHz with 4GB RAM. We discuss the simulation parameters in the next subsections.

5.1. Size of the banner

Five standard banner sizes that are commonly used in Web advertising [33] have been selected to be used for each of the simulation cycles. Table 1 shows the width $\mathcal{W}$ and the height $\mathcal{H}$ of the banners. 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.

5.2. *Sorting of the advertisements*

For every allocation process we have a list of advertisements, through which it is iterated sequentially. Therefore, the order of the elements in the list influences the generated pattern. We distinguish between a random order, and an order that is obtained by sorting according to specific criteria. We created the following sorting criteria from advertisement parameters: price per advertisement pixel (pp), width (w), height (h), total area ($w \times h$), flatness ($^{w}/_{h}$), and proportionality ($|\log(^{w}/_{h})|$). For the sorting criteria, we allow the sorting of the advertisements in both ascending and descending order. The flatness specifies whether the advertisement is flat (width > height) or tall (height > width). 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 in the positive or negative direction signifies that the rectangle is flat or tall. For advertisements ranked the same by the first sorting we use a secondary sort, based on one of the remaining criteria. Altogether the list of advertisements is sorted in $\frac{12!}{10!} - 12 + 1 = 121$ different ways, with two directions (ascending and descending), excluding the situations where the primary sort equals the secondary sort, and adding one random order.

Table 1: Standard banner sizes

$\mathcal{W} \times \mathcal{H}$	Banner
728×90	Leader board
234×60	Half banner
125×125	Square button
120×600	Skyscraper
336×280	Large rectangle

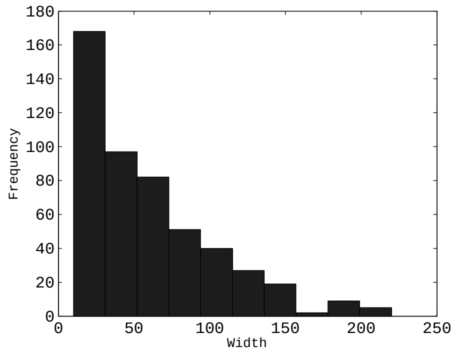

Figure 4: Distribution of advertisement widths for 336×280 banner

5.3. *Size of the advertisements*

For our simulation the advertisements are pseudo-randomly generated. Since the size of the advertisements is related to the size of the banner, we change it for every banner size. In order to achieve 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 independently from each other. The most important constraint was that on the "Million Dollar Homepage" all pixels were sold to advertisers in blocks of 10×10 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 advertisement size of 10×10 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. (6), where the random number is drawn from the standard normal distribution.

$$w_i, h_i = \max\Big(10, \min\Big(\mathcal{W}, \mathcal{H}, \Big\lceil (\min(\mathcal{W}, \mathcal{H})/40) \times |\text{random number}| \Big\rceil \times 10\Big)\Big) \tag{6}$$

To illustrate the distribution this formula generates, a histogram with the widths of a set of 500 ads for a 336×280 banner is displayed in Figure 4. 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 create a number of advertisements such that the total size of the advertisements is approximately twice the size of the banner.

5.4. *Price of the advertisements*

We measure the price for advertisements in price per pixel, this means the prices of the advertisements are proportional to their dimensions. The price per pixel can differ between advertisements due to negotiations with the owner of the banner. The price per pixel for an advertisement is set to 10 added with a random one decimal value uniformly distributed between -1 and 1 ($-1.0, -0.9, -0.8, ..., 0.8, 0.9, 1.0$), resulting in a uniform distribution between 9.0 and 11.0 with a step of 0.1 ($9.0, 9.1, 9.2, ..., 10.8, 10.9, 11.0$). The price of the advertisement is calculated by multiplying this price per pixel with its total area. The larger the total area of the advertisement, the higher the revenue for the owner of the banner.

6. Simulation results

In this section we present and analyse the results of our simulations. First, we present the results of the simulations focused on finding the optimal solution in Section 6.1. These simulations are run on a decreased problem size. Second, we present the results of the heuristic algorithm benchmark in Section 6.2, that uses the experimental design as presented in Section 5.

6.1. Optimal solution benchmark

In this benchmark, we simulate the heuristic algorithms and the brute force search algorithm, with the purpose to gain insight into the effectiveness and efficiency of these algorithms. The enumerative brute force search algorithm always finds the optimal solution, yet finding the optimal solution is extremely time consuming. To overcome this problem, we decrease the problem size for this particular benchmark and use four instances based on the following variables. Instead of the banner sizes present in Table 1, we use only two small banners of $\mathcal{W} = 4$ and $\mathcal{H} = 4$, and $\mathcal{W} = 5$ and $\mathcal{H} = 4$. Instead of the advertisements generated by Eq. 6, we use two small sets of advertisements, $\mathcal{A}_1$ and $\mathcal{A}_2$, which are presented in Table 2 and Table 3, respectively. We provide these four instances to emphasize the enormous increase in computation time when using a larger set of advertisements or an increased banner size.

Table 2: $\mathcal{A}_1$ advertisements

i	w_i	h_i	pp_i
1	1	1	9.1
2	2	3	9.3
3	1	2	9.5
4	1	1	9.7
5	3	2	9.9
6	2	1	10.1
7	1	1	10.3
8	2	2	10.5
9	3	1	10.7
10	1	3	10.9

Table 3: $\mathcal{A}_2$ advertisements

i	w_i	h_i	pp_i
1	1	1	9.0
2	2	3	9.2
3	1	2	9.4
4	1	1	9.6
5	3	2	9.8
6	2	1	10.0
7	1	1	10.2
8	2	2	10.4
9	3	1	10.6
10	1	3	10.8
11	1	1	11.0

For the heuristics we simulated all possible combinations of the sorting criteria. The brute force search algorithm does not use sorting criteria for the advertisements. In Table 4, the best price per pixel profits for each algorithm per instance are shown, wherein $\mathcal{A}$ represents the set of advertisements, s_1 represents the primary sorting criteria, s_2 represents the secondary sorting criteria, *Asc.* means ascending, and *Desc.* means a descending sorting order. One of the optimal solutions for each instance corresponding to the brute force search simulations are showed in Figure 5. Despite the decreased problem size, the brute force search algorithm finished with the smallest instance after 3 minutes, while the largest instance finished after 7 hours on an Intel Core 2 Quad CPU at 2.40 GHz with 4GB RAM. As the results show, increasing the problem size increases the execution time of finding the optimal solution exponentially. This emphasizes the difficulty in finding the optimal solution in a relatively small amount of time. Running the brute force search algorithm for a banner size from Table 1 takes too much time, and justifies the use of heuristic algorithms for multiple advertisement allocation, especially for application on the Web.

For most instances, the heuristic algorithms are able to find the optimal or a near-optimal solution, except for the greedy stripping algorithm. The latter algorithm is the only one that creates waste, i.e., non-allocated space, due to the nature of how the algorithm builds a solution. The results show that, apart from their impressive efficiency, the effectiveness of the left justified algorithm, the orthogonal algorithm, and the GRASP constructive algorithm is high as well. However, the problem size is too small to draw generally valid conclusions with respect to the effectiveness. Nevertheless, the previous experiment – despite being performed on a small data set – gives an idea of the results (quality of the solution and the execution time required to find the solution) obtained with brute force search, and the differences with respect to the results obtained using heuristics.

Table 4: Best profits per pixel for each algorithm per instance

Instance	$\mathcal{W} \times \mathcal{H}$	$\mathcal{A}$	*Algorithm*	s_1	s_2	*Profit*	*Waste*	*Exec. time* (s)
(a)	4×4	$\mathcal{A}_1$	Brute force search	–	–	166.00	0%	186.963732
			Left justified	Proportionality Desc.	Price/pixel Desc.	166.00	0%	0.000200
			Orthogonal	Proportionality Desc.	Price/pixel Desc.	166.00	0%	0.000374
			Orthogonal	Price/pixel Desc.	–	166.00	0%	0.000434
			GRASP con.	Proportionality Desc.	Price/pixel Desc.	166.00	0%	0.001007
			Greedy str.	Width Desc.	Price/pixel Desc.	120.80	25%	0.000353
			Greedy str.	Total area Desc.	Price/pixel Desc.	120.80	25%	0.000396
(b)		$\mathcal{A}_2$	Brute force search	–	–	165.80	0%	1285.800622
			Left justified	Price/pixel Desc.	–	165.60	0%	0.000211
			Orthogonal	Price/pixel Desc.	–	165.60	0%	0.000451
			GRASP con.	Proportionality Desc.	Price/pixel Desc.	165.80	0%	0.001217
			Greedy str.	Width Desc.	Price/pixel Desc.	120.40	25%	0.000344
			Greedy str.	Total area Desc.	Price/pixel Desc.	120.40	25%	0.000364
(c)	5×4	$\mathcal{A}_1$	Brute force search	–	–	206.40	0%	2496.981999
			Left justified	Shape Desc.	$*$	201.80	0%	0.000197
			Orthogonal	Shape Desc.	$*$	201.80	0%	0.000440
			GRASP con.	Height Desc.	Price/pixel Desc.	202.80	0%	0.000990
			Greedy str.	Height Desc.	$*$	156.80	20%	0.000480
			Greedy str.	Price/pixel Asc.	$*$	156.80	20%	0.000432
			Greedy str.	Shape Asc.	$*$	156.80	20%	0.000444
(d)		$\mathcal{A}_2$	Brute force search	–	–	205.80	0%	21533.128504
			Left justified	Width Desc.	Price/pixel Desc.	201.80	0%	0.000279
			Orthogonal	Width Desc.	Price/pixel Desc.	201.80	0%	0.000511
			GRASP con.	Height Desc.	Price/pixel Desc.	202.20	0%	0.001342
			Greedy str.	Height Desc.	Price/pixel Desc.	157.20	20%	0.000468
			Greedy str.	Shape Asc.	Price/pixel Desc.	157.20	20%	0.000458

– = sorting doesn't influence order
$*$ = any sorting

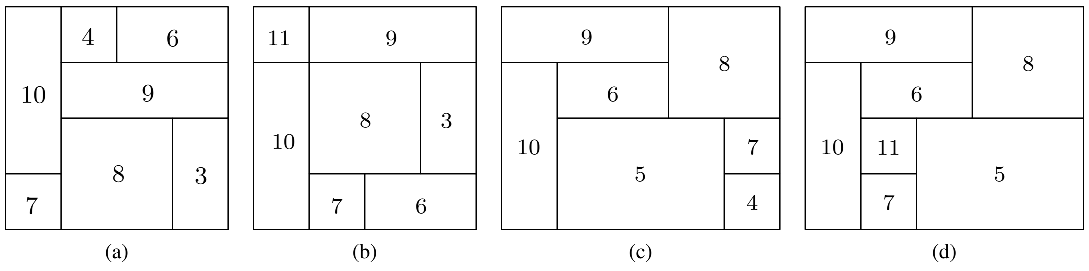

Figure 5: Optimal solution per instance

6.2. Heuristic algorithms benchmark

In this simulation we compare the heuristics presented in Section 4 with respect to effectiveness and efficiency. The configuration of this simulation is set out in Section 5. To summarize, the configuration parameters consist of 9 different banner sizes, 121 different sorts of the set of advertisements, and 4 algorithms, which altogether result in 4356 simulation cycles. We run the full experiment 10 times to rule out bias following from the set of advertisements. The results presented in this section are, unless stated differently, averaged over the ten instances. To avoid bias towards particular banner sizes, we normalize the profit of the banner. We define the profit per banner pixel $PP = \frac{P_{total}}{\mathcal{W} \times \mathcal{H}}$, wherein P_{total} is the total profit of the allocated pattern. Similarly, we define the execution time per banner pixel $TP = \frac{T_{total}}{\mathcal{W} \times \mathcal{H}}$, wherein T_{total} is the total execution time for the allocated pattern.

Since the same set of advertisements is used for all heuristic algorithms within an instance, we can evaluate their performance by comparing the averaged normalized profits and execution times. We first discuss the overall performance, and then cover the performance for specific sorting criteria and banner sizes.

6.2.1. Overall performance

For the overall performance we consider the full data set that was obtained from all instances, without prespecifying any banner size or sorting criteria. The profits per banner pixel are aggregated – average for different banner sizes and sorting criteria – per algorithm. The mean and a five point summary of the distribution of PP for each algorithm is shown in Table 5.

Table 5: Five point summary and mean of the profit per banner pixel (PP), waste rate, and execution time (T_{total})

Profit per banner pixel (PP) per algorithm

Algorithm	*Minimum*	*Q1*	*Median*	*Mean*	*Q3*	*Maximum*
Left justified	6.6328	9.0131	9.5442	9.3510	9.8518	10.5070
Orthogonal	6.6041	9.1042	9.6028	9.3774	9.8686	10.5150
GRASP con.	5.4448	8.6981	9.4207	9.1317	9.8205	10.4700
Greedy str.	5.2888	8.7402	9.3102	9.0821	9.6949	10.3820

Waste rate per algorithm

Algorithm	*Minimum*	*Q1*	*Median*	*Mean*	*Q3*	*Maximum*
Left justified	0.00000	0.01633	0.03460	0.06586	0.09805	0.33955
Orthogonal	0.00000	0.01462	0.02902	0.06323	0.09007	0.33843
GRASP con.	0.00000	0.01709	0.05467	0.08778	0.13029	0.45663
Greedy str.	0.01099	0.02760	0.06660	0.09299	0.12789	0.47696

Execution time (T_{total}) per algorithm in seconds

Algorithm	*Minimum*	*Q1*	*Median*	*Mean*	*Q3*	*Maximum*
Left justified	0.04568	0.17968	1.46220	2.07490	2.86220	14.02800
Orthogonal	0.04939	0.19310	1.82130	2.05550	2.99550	10.90480
GRASP con.	0.00565	0.01508	0.05819	0.08005	0.13044	0.31949
Greedy str.	0.00129	0.00277	0.00363	0.00613	0.00879	0.02787

We can note that overall, the orthogonal algorithm most often yields the highest profit per banner pixel, followed by the left justified algorithm. The greedy stripping algorithm is the least effective. We define the waste rate as the ratio of unallocated pixels to 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. The obtained correlation value of -0.9767 shows that a lower waste rate will result in a higher profit per banner pixel. This is confirmed by the distribution of the waste rate per algorithm, which is also presented in Table 5. The orthogonal algorithm produces the least waste, whereas the greedy stripping algorithm on average produces the most waste.

The efficiency of the algorithms, i.e., the execution time in seconds, is also presented in Table 5. With respect to efficiency, the left justified algorithm and the orthogonal algorithm perform the worst. 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. The latter two algorithms loose a lot of time by iterating through the banner for every advertisement.

To determine whether the results are significant we performed one-tailed paired t-tests for the profit per banner pixel and the execution time, based on the means of the ten instances given in Table 6. The significance tests are performed with $\alpha = 0.05$. The orthogonal algorithm is significantly more effective than the left justified algorithm. Furthermore, the left justified algorithm is significantly more effective than the GRASP constructive algorithm. Last, the GRASP constructive algorithm is significantly more effective than the greedy stripping algorithm. With respect to the effectiveness, the greedy stripping algorithm was significantly more efficient than the GRASP constructive algorithm. The GRASP constructive algorithm is more efficient than the left justified algorithm and the orthogonal algorithm, also statistically significant. The mean execution time of the orthogonal algorithm is lower than the left justified algorithm, but this difference is not significant. Therefore, we can conclude that the left justified algorithm and the orthogonal algorithm perform similarly with respect to efficiency.

Table 6: Mean profit per banner pixel (PP) and execution time (T_{total}) per algorithm per instance

Mean profit per banner pixel (PP) per algorithm per instance

Algorithm	1	2	3	4	5	6	7	8	9	10
Left justified	9.339	9.370	9.416	9.345	9.386	9.352	9.287	9.329	9.334	9.352
Orthogonal	9.383	9.398	9.452	9.367	9.382	9.398	9.334	9.344	9.350	9.366
GRASP con.	9.077	9.136	9.233	9.151	9.177	9.155	9.085	9.098	9.102	9.103
Greedy str.	9.079	9.098	9.111	9.082	9.138	9.107	9.036	9.081	9.047	9.042

Mean execution time (T_{total}) per algorithm per instance in seconds

Algorithm	1	2	3	4	5	6	7	8	9	10
Left justified	2.16700	1.98400	2.12200	2.01200	2.22300	1.89300	2.27100	1.95800	2.12000	1.99900
Orthogonal	2.09400	1.97100	2.06200	2.04900	2.30800	1.96500	2.20800	2.00000	1.92000	1.97800
GRASP con.	0.07871	0.07800	0.08096	0.07914	0.08242	0.08196	0.08176	0.07591	0.08385	0.07779
Greedy str.	0.00615	0.00611	0.00618	0.00597	0.00640	0.00604	0.00613	0.00586	0.00639	0.00601

Table 7: Execution time and profit per banner pixel for all primary sorts for each algorithm

Algorithm	*Primary sort*	T_{total} (s)	PP
Orthogonal	Price/pixel Desc.	1.75	10.03
Left justified	Price/pixel Desc.	1.83	10.02
Orthogonal	Total area Desc.	2.30	9.80
Left justified	Total area Desc.	2.10	9.79
GRASP con.	Total area Desc.	0.05	9.76
Left justified	Width Desc.	1.23	9.75
GRASP con.	Price/pixel Desc.	0.07	9.74
Orthogonal	Height Desc.	1.22	9.71
Orthogonal	Width Desc.	1.29	9.70
Greedy str.	Price/pixel Desc.	0.01	9.70
Left justified	Height Desc.	1.53	9.68
GRASP con.	Height Desc.	0.07	9.65
GRASP con.	Width Desc.	0.06	9.63
Orthogonal	Random order	1.63	9.62
Left justified	Random order	1.82	9.61
Left justified	Proportionality Asc.	1.93	9.59
Orthogonal	Proportionality Asc.	2.03	9.58
GRASP con.	Proportionality Asc.	0.08	9.48
Greedy str.	Proportionality Asc.	0.01	9.46
Left justified	Flatness Desc.	2.14	9.43
Greedy str.	Total area Desc.	0.00	9.42
Orthogonal	Flatness Asc.	1.98	9.42
Orthogonal	Proportionality Desc.	1.97	9.39
Orthogonal	Flatness Desc.	2.07	9.38
Greedy str.	Random order	0.01	9.36
GRASP con.	Random order	0.07	9.34
Greedy str.	Width Desc.	0.00	9.34
Left justified	Proportionality Desc.	2.26	9.33
Left justified	Flatness Asc.	1.54	9.31
GRASP con.	Flatness Asc.	0.09	9.23
Greedy str.	Height Desc.	0.00	9.22
GRASP con.	Flatness Desc.	0.08	9.14
Left justified	Price/pixel Asc.	1.87	9.12
Greedy str.	Flatness Desc.	0.01	9.11
Orthogonal	Price/pixel Asc.	1.79	9.10
Greedy str.	Flatness Asc.	0.01	9.07
Greedy str.	Proportionality Desc.	0.01	9.03
Greedy str.	Price/pixel Asc.	0.01	8.99
GRASP con.	Proportionality Desc.	0.09	8.97
GRASP con.	Price/pixel Asc.	0.08	8.91
Orthogonal	Height Asc.	2.67	8.89
Orthogonal	Width Asc.	2.66	8.87
Left justified	Height Asc.	3.66	8.82
Left justified	Width Asc.	1.75	8.74
Orthogonal	Total area Asc.	2.99	8.65
Greedy str.	Total area Asc.	0.01	8.62
Left justified	Total area Asc.	3.10	8.61
Greedy str.	Width Asc.	0.01	8.51
Greedy str.	Height Asc.	0.01	8.50
GRASP con.	Height Asc.	0.09	8.43
GRASP con.	Width Asc.	0.10	8.38
GRASP con.	Total area Asc.	0.11	8.23

6.2.2. Sorting

The preliminary sorting of the advertisements influences the final allocation pattern, since all heuristic algorithms iterate through the sorted list of advertisements. In Table 7, the price per pixel and execution time is given for all 12 primary sorting criteria, for each algorithm. Note that we still have aggregated the banner sizes and secondary sorting criteria, nevertheless, the influence of the secondary sort on the effectiveness is relatively small. 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 criteria. After that, sorting the set of advertisements descending according to total area, yields the best profit per banner pixel for the orthogonal algorithm and left justified algorithm. Moreover, the GRASP constructive algorithm is most effective with the descending total area sorting criteria. Based on these results, we can assume that in general it is better to place larger advertisements first, and then filling up the empty spaces with smaller ones. Apart from the sorting criteria, we also considered a random sorting order. From the results we can conclude that the random sorting order has a moderate performance with respect to the profit per banner pixel. This justifies the use of specific sorting criteria to increase the effectiveness.

In the previous subsection we already noticed the greedy stripping algorithm outperforming the other algorithms when it comes to execution time. The left justified algorithm and orthogonal algorithm have the lowest efficiency, and the GRASP constructive algorithm is somewhere in the middle. An interesting observation is that sorting the set of advertisements according to the descending dimension criteria, i.e., total area, width, height, is more efficient than sorting them in an ascending way. An explanation is that placing larger advertisements first yields a lower number of allocated ads, and shortens the execution time. Although the orthogonal algorithm is most effective, it is least efficient, together with the left justified algorithm. For our purposes of using the algorithm on the Web, these algorithms are still far below the specified maximum execution time of 30 seconds. In this case, we prefer effectiveness over efficiency, but in other cases the need for efficiency might be higher, and the greedy stripping and GRASP constructive algorithm are more suitable.

6.2.3. Banner size

In this section we specify for each banner size, for every algorithm, the best setting for the sorting criteria in order to maximize revenue. Recall that we considered the five standard banner sizes 'leader board', 'half banner', 'square button', 'skyscraper' and, 'large rectangle' [33]. The most effective algorithm settings per banner size are given in Table 8.

Table 8: Most effective algorithm settings per banner size

Banner size	*Algorithm*	*Primary sort*	*Secondary sort*	*TP (s)*	*PP*
728 × 90	Left justified	Price/pixel Desc.	Total area Desc.	0.00002644	10.4004
(*Leader board*)	Orthogonal	Price/pixel Desc.	Total area Desc.	0.00002510	10.3997
	Greedy str.	Price/pixel Desc.	Flatness Asc.	0.00000016	10.2806
	GRASP con.	Price/pixel Desc.	Width Desc.	0.00000253	10.0986
234 × 60	Left justified	Price/pixel Desc.	Total area Desc.	0.00000880	10.3150
(*Half banner*)	Orthogonal	Price/pixel Desc.	Height Desc.	0.00000966	10.3145
	GRASP con.	Price/pixel Desc.	Width Desc.	0.00000101	10.2446
	Greedy str.	Price/pixel Desc.	Height Desc.	0.00000020	10.1122
125 × 125	Left justified	Price/pixel Desc.	Total area Desc.	0.00001133	9.5060
(*Square button*)	GRASP con.	Price/pixel Desc.	Width Desc.	0.00000060	9.4337
	Orthogonal	Price/pixel Desc.	Flatness Asc.	0.00001008	9.3644
	Greedy str.	Width Desc.	Price/pixel Desc.	0.00000012	9.0669
120 × 600	Orthogonal	Price/pixel Desc.	Total area Desc.	0.00003000	10.4337
(*Skyscraper*)	GRASP con.	Price/pixel Desc.	Total area Desc.	0.00000136	10.4151
	Left justified	Price/pixel Desc.	Total area Desc.	0.00001948	10.3181
	Greedy str.	Price/pixel Desc.	Width Desc.	0.00000011	10.0265
336 × 280	Left justified	Total area Desc.	Price/pixel Desc.	0.00002650	9.6746
(*Large rectangle*)	Orthogonal	Total area Desc.	Proportionality Asc.	0.00002482	9.6636
	GRASP con.	Total area Desc.	Price/pixel Desc.	0.00000032	9.5713
	Greedy str.	Height Desc.	Proportionality Asc.	0.00000003	9.1050

Leader board 728 x 90. All algorithms are most effective using a descending price per pixel as the primary sorting criteria. The left justified algorithm performs best on the leader board, followed by the orthogonal algorithm, with the same secondary sorting criteria, i.e., a descending total area. The two algorithms perform almost similar with respect to both effectiveness and efficiency. The greedy stripping algorithm outperforms the GRASP constructive algorithm. The latter algorithm performs best when sorting according to a descending width, but is the least effective for this banner size.

Half banner 234 x 60. Again, the most effective primary sorting order is a descending price per pixel for all algorithms. The left justified algorithm performs best, using a descending total area as secondary sorting criteria. The orthogonal algorithm performs almost similar, but with a descending height as secondary sort. For the half banner, the GRASP constructive algorithm outperforms the greedy stripping algorithm.

Square button 125 x 125. For this square shaped banner the left justified algorithm is most effective, again using a descending total area as secondary sorting criteria. The GRASP constructive algorithm outperforms the orthogonal algorithm, which performs very poor with this banner size. The greedy stripping algorithm performs best when using a descending width as primary sorting criteria.

Skyscraper 120 x 600. All algorithms perform best when using a descending price per pixel primary sorting. The orthogonal algorithm with a descending total area secondary sorting is most effective for the skyscraper. For this banner size, the GRASP constructive algorithm beats the left justified algorithm. The three algorithms perform best using the same secondary sorting criteria.

Large rectangle 336 x 280. For the large rectangle, the left justified algorithm is most effective, using a descending total area primary sorting and a descending price per pixel secondary sorting. The orthogonal algorithm performs best when using the ascending proportionality secondary sorting criteria. The greedy stripping algorithm performs best with the descending height primary sorting criteria.

Overall, the skyscraper yields the most profit per banner pixel, followed by the leader board and half banner. Note that similar conclusions cannot be drawn about the total revenue the banner creates, since the banners have different sizes. Of course, using the large rectangle banner creates more revenue than the other banners. Regarding the execution times per banner pixel we observe that the half banner, followed by the square button banner are the most efficient. Again similar conclusions cannot be be drawn about the total execution time for a banner, as the banners have different sizes.

7. Conclusions and future work

In this paper, we have presented heuristic-based solutions for a modified version of the pixel advertisement problem. We focused on allocating multiple advertisements on a banner and proposed several heuristic algorithms that provide adequate solutions for the problem. Our experiments give insight into the effectiveness and efficiency of these algorithms. The orthogonal algorithm is the most effective, followed by the left justified algorithm. In contrast, the greedy stripping algorithm is the most efficient, followed by the GRASP constructive algorithm. When we introduce a specific primary sorting for the set of advertisements, the orthogonal algorithm using a descending price per pixel is the most effective. Moreover, sorting the set of advertisements according to a descending price per pixel also yields the best profit per banner pixel for the left justified and the greedy stripping algorithm. Sorting by a descending total area is the best for the GRASP constructive algorithm. The descending total area primary sorting order performs second best after the descending price per pixel for the orthogonal algorithm and the left justified algorithm. Based on these results, we can assume that in general it is better to place larger advertisements first, and then filling up the empty spaces with smaller ones. We also specified the most effective settings for each algorithm for every banner type used. Overall, the skyscraper yields the most profit per banner pixel, followed by the leader board and half banner.

This research also uncovers directions for future work. At the current moment our research is limited to the allocation algorithms that we have defined, while other heuristics may yield a better effectiveness-efficiency tradeoff. First, the left justified and orthogonal algorithm can be improved. These algorithms are quite straightforward in that they iterate through the banner every time, revisiting places that are known to be occupied. Saving this information and thereby preventing revisits could make the algorithms more efficient. Second, the heuristic algorithms might be extended. An idea is to reshuffle the placed advertisements in the banner with the purpose to create more contiguous empty space, or in such a way that the next advertisement can be placed while obeying the previous constraint. Third, besides the use of heuristic algorithms as we present in the paper, one could apply metaheuristics, e.g., a genetic algorithm, to optimally place advertisements in the banner.

It might be more realistic to assign different prices to particular positions on the banner. In our research, we have a predefined set of advertisements with predefined prices, regardless of the position they are placed. The Eyetrack III [35] research investigates the movements of the human eye when looking at Web pages. More frequently watched areas in the banner may be assigned a higher price per pixel.

Another idea to explore is to incorporate the semantics of advertisements into our problem definition. We could add semantic constraints to the banner to prevent certain advertisements to be placed side by side. An example is to prevent a Coca-cola advertisement to be allocated besides a Pepsi advertisement. Semantics can also be used to match the advertisements on the banner to the content of the Web page, similar to what Google AdSense [36] does with normal banners.

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 thus increases user attention.

References

[1] Interactive Advertising Bureau. Internet advertising revenue report 2008, `http://www.iab.net/insights/_research/530422/adrevenuereport`.

[2] Tew A. Million Dollar Homepage, `http://www.milliondollarhomepage.com/`.

[3] Wojciechowski A. An improved Web system for pixel advertising. In: Bauknecht K, Pröll B, Werthner H (Eds.). E-commerce and Web technologies. LNCS, vol. 4082. Heidelberg: Springer, 2006. p. 232-241.

[4] Wäscher G, Haußner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2007;183(3):1109-1130.

[5] Hadjiconstantinou E, Christofides N. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. European Journal of Operational Research 1995;83:39-56.

[6] Garey MR, Johnson DS. Computers and intractability; a guide to the theory of NP-completeness. New York: W. H. Freeman & Co., 1990.

[7] Knoops A, Boskamp V, Wojciechowski A, Frasincar F. Single pattern generating heuristics for pixel advertisements. In: Vossen G, Long DDE, Xu Yu J (Eds.). Web Information Systems Engineering. LNCS, vol. 5802. Heidelberg: Springer, 2009. p. 415-428.

[8] Wojciechowski A, Kapral D. Allocation of multiple advertisement on limited space: heuristic approach. In: Mauthe A, Zeadally S, Cerqueira E, Curado M (Eds.). Future Multimedia Networking. LNCS, vol. 5630. Heidelberg: Springer, 2009. p. 230-235.

[9] Adler M, Gibbons PB, Matias Y. Scheduling space-sharing for internet advertising. Journal of Scheduling 2002;5(2):103-119.

[10] Dawande M, Kumar S, Sriskandarajah C. Performance bounds of algorithms for scheduling advertisements on a Web page. Journal of Scheduling 2003;6(4):373-394.

[11] Freund A, Naor JS. Approximating the advertisement placement problem. Journal of Scheduling 2004;7(5):365-374.

[12] Dawande M, Kumar S, Sriskandarajah C. Scheduling Web advertisements: a note on the minspace problem. Journal of Scheduling 2005;8(1):97-106.

[13] Menon S, Amiri A. Scheduling banner advertisements on the Web. INFORMS Journal on Computing 2004;16(1):95-105.

[14] Amiri A, Menon S. Efficient scheduling of internet banner advertisements. ACM Transactions on Internet Technology 2003;3(4):334-346.

[15] Kumar S, Jacob VS, Sriskandarajah C. Scheduling advertisements on a Web page to maximize revenue. European Journal of Operational Research 2006;173(3):1067-1089.

[16] Caprara A, Monaci M. On the two-dimensional knapsack problem. Operations Research Letters 2004;32(1):5-14.

[17] Lodi A, Monaci M. Integer linear programming models for 2-staged two-dimensional knapsack problems. Mathematical Programming 2003;94(2-3):257-278.

[18] Clautiaux F, Carlier J, Moukrim A. A new exact method for the two-dimensional orthogonal packing problem. European Journal of Operational Research 2007;183(3):1196-1211.

[19] Fekete SP, Schepers J, van der Veen J. An exact algorithm for higher-dimensional orthogonal packing. Operations Research 2007;55(3)569-587.

[20] Beasley JE. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 1985;33(1):49-64.

[21] Suliman SMA. A sequential heuristic procedure for the two-dimensional cutting-stock problem. International Journal of Production Economics 2006;99:177-185.

[22] Egeblad J, Pisinger D. Heuristic approaches for the two- and three-dimensional knapsack packing problem. Computers & Operations Research 2009;36(4):1026-1049.

[23] Fayard D, Zissimopoulos V. An approximation algorithm for solving unconstrained two-dimensional knapsack problems. European Journal of Operational Research 1995;84(3):618-632.

[24] Hifi M, Michrafy M, Sbihi A. Heuristic algorithms for the multiple-choice multidimensional knapsack problem. The Journal of the Operational Research Society 2004;55(12):1323-1332.

[25] Gonçalves JF. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. European Journal of Operational Research 2007;183(3):1212-1229.

[26] Lodi A, Martello S, Vigo D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing 1999;11(4):345-357.

[27] Lodi A, Martello S, Monaci M. Two-dimensional packing problems: a survey. European Journal of Operational Research 2002;141(2):241-252.

[28] Lai KK, Chan JWM. An evolutionary algorithm for the rectangular cutting stock problem. International Journal of Industrial Engineering 1997;4:130-139.

[29] Beasley JE. A population heuristic for constrained two-dimensional non-guillotine cutting. European Journal of Operational Research 2004;156(3):601-627.

[30] Lai KK, Chan JWM. Developing a simulated annealing algorithm for the cutting stock problem. Computers & Industrial Engineering 1997;32(1):115-127.

[31] Alvarez-Valdes R, Parreño F, Tamarit JM. A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. The Journal of the Operational Research Society 2005;56(4):414-425.
[32] Alvarez-Valdes R, Parreño F, Tamarit JM. A tabu search algorithm for a two-dimensional non-guillotine cutting problem. European Journal of Operational Research 2007;183(3):1167-1182.
[33] Interactive Advertising Bureau. Ad unit guidelines, http://www.iab.net/iab/_products/_and/_industry/_services/1421/1443/1452.
[34] Korf RE. Depth-first iterative-deepening: an optimal admissible tree search. Artificial intelligence 1985;27(1):97-109.
[35] Outing S, Ruel L. The best of eyetrack III: what we saw when we looked through their eyes. Poynter Institute, http://poynterextra.org/eyetrack2004/main.htm.
[36] Google Inc. Google advertising programs, http://www.google.com/ads/