






# Solving CSS-Sprite Packing Problem Using a Transformation to the Probabilistic Non-oriented Bin Packing Problem

Soumaya Sassi Mahfoudh<sup>(✉)</sup>, Monia Bellalouna, and Leila Horchani

Laboratory CRISTAL-GRIFT, National School of Computer Science,  
University of Manouba, Manouba, Tunisia  
soumaya.lsm@gmail.com, monia.bellalouna@gmail.com,  
leila.horchani@gmail.com

**Abstract.** CSS-Sprite is a technique of regrouping small images of a web page, called tiles, into images called sprites in order to reduce network transfer time. CSS-sprite packing problem is considered as an optimization problem. We approach it as a probabilistic non-oriented two-dimensional bin packing problem ( $2PBPP|R$ ). Our main contribution is to allow tiles rotation while packing them in sprites. An experimental study evaluated our solution, which outperforms current solutions.

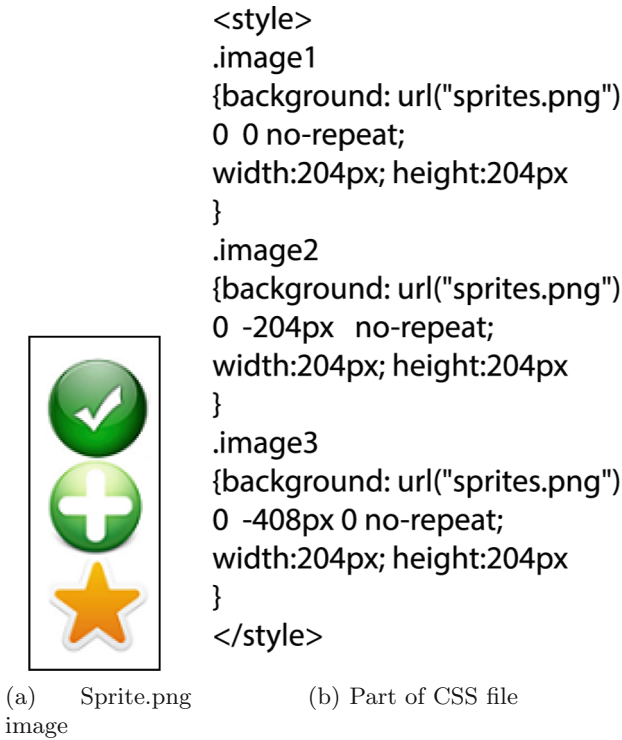
**Keywords:** Bin packing · Non-oriented · CSS-sprite  
Image compression

## 1 Introduction

It was reported in [16] that 61.3% of all HTTP requests to servers are images. In fact, for each image we need a HTTP request. This action includes interaction between the web server and the user. Web server is characterized by a long delay due to the messages transporting the request through the network stack, the request treatment at the server and the location of the resources in the server cache. So to reduce web interactions, web designers resort to CSS-sprite technique, whose main idea is to regroup small images, called tiles, in pictures called, sprites.

Figure 1(a) shows a sprite and Fig. 1(b) shows a part of Cascading Style Sheet (CSS) [27] file. The size of each of the three tiles in Fig. 1(a) is 17 Kilobytes (KB). If tiles are used separately, we need to load each tile apart, which means that we are going to load 51 KB. However, if we use the sprite Fig. 1(a), we need only to load 21 KB. And this is not all, for in order to load each tile, we need a HTTP request instead of loading the sprite only once and saving it on the cache. We can imagine the amount of reduction in the case of thousands of tiles.

To our knowledge, CSS was introduced by [1] then popularized by [23]. CSS-sprite generators pack all tiles in one or multiple sprites. Yet, they are still forcing the packing of tiles without rotation.



**Fig. 1.** Example of use of CSS-sprite

Css-sprite problem is a practical problem with multiple facets involving combinatorial optimization problems, image compression and network performance. These facets will be presented in further sections. In the next section, we will present our approach which allows tiles rotation while constructing sprites. In Sect. 3, we will present in details geometric packing as well as chosen heuristics. In Sect. 4, we will describe briefly image processing. Section 5 is dedicated to outline communication performance. The last section is devoted to the evaluation of our solution.

## 2 Problem Formulation

Formally, CSS-sprite packing problem is defined as follows: given a set of tiles  $\Gamma_n = \{t_1, \dots, t_n\}$  in standard formats (such as JPEG, PNG and GIF). We intend to combine them into a sprite or a set of sprites  $S$  to minimize network transfer time. CSS-sprite packing is a NP-Hard problem [20]. The major problem is the large number of tiles and the presence of distorted tiles. Ccss-sprite packing is

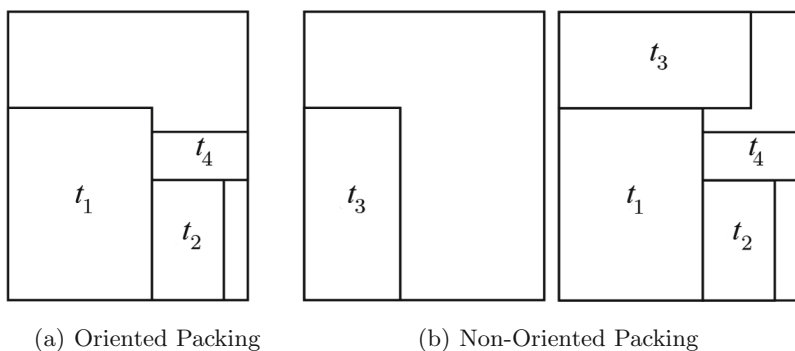
considered as an optimization problem of the class 2D packing problems because tiles and sprites are rectangles. Contemporary CSS-sprite generators pack tiles in one or many sprites but do not consider two important aspects:

1. Tiles rotation.
2. The presence of distorted tiles.

In fact, though it is technically possible to rotate images using CSS, tiles rotation has not been used in CSS-sprite packing so far [20], which may cause wasted space illustrated in Fig. 2. Wasted space drains memory and excessive memory usage affects browser performance. One possible approach to overcome wasted space in sprites is to model CSS-sprite problem as a two-dimensional probabilistic non-oriented bin packing problem. Following the notation [18], this problem is denoted by 2PBPP|R. 2PBPP|R is a branch of Probabilistic Combinatorial Optimization Problems (PCOP).

The idea of PCOPs comes from Jaillet [14,15]. Among several motivations PCOPs were introduced to formulate and analyze models which are more appropriate for real world problems. PBPP was first studied in [5].

2PBPP|R is essentially a 2BPP|R where one is asked to pack a varying number of rectangular items: where we assume that a list  $L_n$  of  $n$  rectangular items is given, and that some items disappear from  $L_n$ . The subset of present items is packed without overlapping and with the possibility of rotation by  $90^\circ$  into the minimum number of identical bins. Table 1 represents the similarities between 2PBPP|R and CSS-sprite problem.



**Fig. 2.** Example of wasted space

Solving CSS-sprite problem, is a tantamount to solving an instance of 2PBPP|R. The possible optimization methods to solve bin-packing problems are exact methods, heuristics, meta-heuristics. Even though, it is guaranteed that exact methods can find an optimal solution, the difficulty of obtaining an optimal solution increases drastically if the problem size increases, due to the fact that is an NP-hard problem.

**Table 1.** Analogy between 2PPBP|R and CSS-sprite technique

<i>2PBPP R</i>	CSS-sprite
$L_n$ : set of rectangular items	$\Gamma_n$ : set of tiles
Bins with same capacity	Sprites with same size
Rectangular items	Tiles
Items rotation $90^\circ$	Tiles rotation $90^\circ$
Absent items	Distorted, unused tiles
Minimize the average number of bins	Better fulfil sprites

### 3 Geometric Packing

Css-sprite packing was firstly solved manually [23] then multiple solutions were proposed. Moreover, a great number of sprite generators have been proposed. A recent survey of existing solutions were proposed by [20]. But we are only interested in those which exploit 2D packing heuristics. Table 2 groups this category of solutions identified by short name and web address. In fact, in CSS-sprite packing problem, decisions of choosing the position of tiles need to be made without full knowledge of the rest of the input. We have an incrementally appearing input, where the input needs to be processed in the order in which it comes. The input is only completely known at the end of the problem.

So, to solve this situation we consider some fast online algorithms. Such algorithms receive the tiles one at a time and need to decide where to place tiles in the bin without knowing the full problem. We choose from literature the following algorithms:

1. Bottom Left (BL): The heuristic was proposed by Baker et al. [4]. The current item is then packed in the lowest position of open bin, left justified; if no bin can allocate it, a new one is initialized. Chazelle [6] proposed an efficient implementation of this algorithm in  $\mathcal{O}(n^2)$  time and  $\mathcal{O}(n)$  space.
2. Best Area Fit (BAF): Orient and place each rectangle to the position where the y-coordinate of the top side of the rectangle is the smallest and if there are several such valid positions, pick the one that is smallest in area to place the next item into. The item is placed in the bottom left corner of the chosen area. Based on tests performed by [2], it would suggest an average of  $\mathcal{O}(n^3)$  time and  $\mathcal{O}(n)$  space.
3. Item Maxim Area (IMA): This heuristic was proposed by [9] as an extension of the Best-Fit heuristic for 2D packing problems. At each step of item packing, a choice of the couple (item to be packed, receiving area) is made. This choice is based on the criteria which takes into account the characteristics of the item and those of the candidate area. Given an item  $a_i(w_i, h_i)$  in a given orientation and an area  $ma$  that can contain it, let  $dx_i$  and  $dy_i$  (respectively  $w_{ma}$  and  $h_{ma}$ ) be the projections of the edges of  $a_i$  (respectively  $ma$ ) on the  $x$ - and  $y$ -axis. Given four real numbers:  $q_1, q_2, q_3$  and  $q_4$  such that  $0 \leq q_k \leq 1; k = 1, \dots, 4$  and  $\sum_{k=1, \dots, 4} q_k = 1$ , the criteria can be written as follows:

**Table 2.** Sprite generators using 2D packing algorithms

	Short name	Output format	2D packing heuristic	Web address
A	Glue	PNG PNG8	Binary-tree [11]	<a href="http://glue.readthedocs.io/en/latest/">http://glue.readthedocs.io/en/latest/</a>
	Zerosprites	PNG PNG8	Korf's algorithm [13] B*-tree [8]	<a href="http://zerosprites.com/">http://zerosprites.com/</a>
	Pypack	PNG	Extension of binary-tree [11]	<a href="http://jwezorek.com/2013/01/sprite-packing-in-python/">http://jwezorek.com/2013/01/sprite-packing-in-python/</a>
	JSGsf	PNG	Binary-tree [11]	<a href="https://github.com/jakesgordon/sprite-factory/">https://github.com/jakesgordon/sprite-factory/</a>
	Isacc	PNG	Rectangle packing [10]	<a href="https://www.codeproject.com/Articles/140251/Image-Sprites-and-CSS-Classes-Creator">https://www.codeproject.com/Articles/140251/Image-Sprites-and-CSS-Classes-Creator</a>
	Simpreal	PNG JPEG GIF BMP	Colum or row mode	<a href="http://simpreal.org.ua/csssprites/#!source">http://simpreal.org.ua/csssprites/#!source</a>
B	Codepen	PNG	Tiles sorting by area width or height	<a href="https://codepen.io/JFarrow/full/scxKd">https://codepen.io/JFarrow/full/scxKd</a>
	Csgencom	PNG JPEG GIF	Not specified	<a href="http://css.spritegen.com/">http://css.spritegen.com/</a>
	Cdplxsg	PNG	Tree [7, 21]	<a href="http://spritegenerator.codeplex.com/">http://spritegenerator.codeplex.com/</a>
	Txturepk	Many formats	MaxRects [3] Bottom-left [4]	<a href="https://www.codeandweb.com/texturepacker/documentation">https://www.codeandweb.com/texturepacker/documentation</a>
	Stitches	PNG	Not specified	<a href="http://draeton.github.io/stitches/">http://draeton.github.io/stitches/</a>
	Sstool	PNG	Not specified	<a href="https://www.leshylabs.com/apps/sstool/">https://www.leshylabs.com/apps/sstool/</a>
	Canvas	PNG	Korf's algorithm [17]	<a href="https://timdream.org/canvas-css-sprites/en/">https://timdream.org/canvas-css-sprites/en/</a>
	Shoebox	PNG	Not specified	<a href="https://renderhjs.net/shoebox/">https://renderhjs.net/shoebox/</a>
	Retina	PNG JPEG GIF	Colum row diagonal mode	<a href="http://www.retinaspritegenerator.com/">http://www.retinaspritegenerator.com/</a>
	Csspg	PNG JPEG GIF	Binary-tree top-down left-right	<a href="https://www.toptal.com/developers/css/sprite-generator">https://www.toptal.com/developers/css/sprite-generator</a>
	Spritepack	PNG8 PNG32 PNG24 JPEG GIF	FFDH [19] BFDH [19] Bottom-left [4]	<a href="http://www.cs.put.poznan.pl/mdrozdowski/spritepack/">http://www.cs.put.poznan.pl/mdrozdowski/spritepack/</a>

$$O(a_i, ma) = q_1 \frac{w_i h_i}{w_{ma} h_{ma}} + q_2 \frac{dx_i}{w_{ma}} + q_3 \frac{dy_i}{h_{ma}} + q_4 \frac{w_i^2 + h_i^2}{w_{ma}^2 + h_{ma}^2}$$

The couple (item to be packed, maximal area that will accommodate it) is the one that maximizes the criteria cited above. The choice of IMA was based on the elaborated experiments [9], which conclude that IMA dominates several heuristics from literature however theoretically the complexity of this heuristic is  $\mathcal{O}(n^5)$ .

## 4 Image Processing

Processing images is a primordial step in CSS-sprite packing whose purpose is to reduce tiles sizes, and so implicitly decrease transfer time and sprites size. It involves tiles transformation and tiles compression.

1. Tiles Transformation: Tiles are images in standard image formats as JPEG, PNG and GIF. All GIF's tiles were converted to PNG, which reduces image size [24]. JPEG tiles were transformed to PNG if PNG format is smaller than JPEG image.
2. Tiles compression: Presenting image compression techniques and standards is beyond the scope of this paper. But we recommend readers to take a look at several survey papers [22, 25] to understand the concept of image compression techniques and standards. In fact, no method can be considered good for all images, nor are all methods equally good for a particular type of image. Compression methods perform in different manner in accordance with different kinds of images.

Recently, Google Incorporation proposed a compression tool named Zopfli [3]. Zopfli algorithm is based on Huffman coding. It was proved that Zopfli yields the best compression ratio [12].

As we mentioned before, images often represent the majority of bytes uploaded to a web page. Therefore, image optimization is essential for saving bytes and the most important performance improvement. For better results, sprites were post-compressed for the minimum size. This means that sprites obtained after packing tiles are further compressed for the minimum size.

## 5 Communication Performance

Obviously, we consider that measuring the quality of sprites is equivalent to determining the network transfer time. However, certain factors make it hardly possible. In fact, transfer time is unpredictable and non-deterministic. So, it remains impossible to use detailed methods of packet level simulation to calculate sprites transfer time since those methods are quite time consumers [20]. Thus, [26] proposed to use flow models to evaluate the quality of sprites.

We exploited the flow model proposed by [20] which was validated in real settings. Table 3 presents the parameters of our model:

**Table 3.** Model parameters

Parameter	Definition
$S$	Set of sprites
$m$	Number of sprites
$f_i$	Size of sprite $S_i$ in bytes
$F$	Size of set $S$
$c$	Number of communication channels
$B(c)$	Accumulated bandwidth of $c$
$L$	Communication latency (startup time)
$T(S, c)$	Transfer time as a function of $S$ and $c$

The transfer time of a set of sprites over  $c$  concurrent channels is modeled by the following formula [20]:

$$T(S, c) = \max \left\{ \frac{1}{c} \sum_{i=1}^m \left( L + \frac{f_i}{B(c)/c} \right), \max_{\{i=1..m\}} \left\{ L + \frac{f_i}{B(c)/c} \right\} \right\} \quad (1)$$

Since the web site performance is not only affected by the server but also by the user side such as browser and computer performance, so performance parameters should be measured on their real populations.

## 6 Computational Results

In this section, we compare our approach to solve CSS-sprite packing problem, named SpriteRotate, with alternative sprite generators. The main contribution of our approach is to rotate tiles by  $90^\circ$  while constructing sprites.

SpriteRotate has been implemented in Java using Eclipse Jee Neon IDE. All tests were performed on typical PC with i5-5200U CPU (2.2 GHz), 12 GB of RAM and Windows 8.

Based on experiments through real visitors [20], transfer time model parameters have been set to  $L = 352$  ms,  $c = 3$  and  $B(c) = 631$  Kilobit (Kb)/s. For image compression, Zopfli compression level has been set to the strongest level 9.

Generated sprites by SpriteRotate include the position of tile in the sprite, which sprite contains a considered tile and whether the output is one sprite or multiple sprites. Besides, we specify if the tile in the sprite is rotated or not to facilitate the extraction of tiles from CSS file. SpriteRotate offers two output formats: PNG and JPEG.

Thereafter, we applied the following procedure. In the first experiments, we considered only a set of sprite generators which construct one sprite. Since SpriteRotate builds a number of sprites, we modified SpriteRotate code to generate a single sprite. In fact, group A of solutions in Table 2 were excluded from

the evaluation because of: failure to work properly or dead applications. Only solutions from group B were chosen for comparison.

In the second series of tests, SpriteRotate has been compared to Spritepack [20], which is a recent solution which generates multiple sprites. The comparison focused on the sizes of the sprites and the objective function: transfer time.

In order to evaluate SpriteRotate, we considered 10 tiles sets from test sets collected in [20]. The tiles are skins and other reusable GUI elements of popular open source web applications. But unfortunately most of them are too simple, consisting of few tiles with identical shape and tiles format. Nevertheless, this tiles test sets allow evaluating our approach in realistic settings. The instances in Table 4 are chosen to represent a spectrum of possible situations: from Joomla Busines14a tile set of size smaller than 20 KB (29 tiles) to Vbulletin Darkness with 1010 tiles and over 11.2 Megabytes (MB) total size.

The results of the first evaluations are collected in Tables 5 and 6, which show the sprite size  $f_i$  and resulted transfer time  $T(S, c)$  of SpriteRotate compared to alternative generators. Each column represents results for each generator. Column labeled “Min” and “Max” represents respectively the minimum and the maximum gain rate obtained by SpriteRotate relatively to alternative generators. Row “Average” is the average size of the sprite through all test instances. An empty cell means that generators has not been able to generate a sprite. It is clear that SpriteRotate outperformed the alternative generators in sprite size and transfer time. Codepen generator considered as the second generator, multiplied on average sprite size by a factor 4 compared to SpriteRotate’s (17 in worst case). Similarly, transfer time was multiplied on average by a factor of 5 compared to the SpriteRotate’s objective function (and 28 in the worst case). In absolute terms, SpriteRotate decreases sprite size from 16 KB to 279 KB. As consequence, a very considerable gain was obtained. SpriteRotate succeed to reduce transfer time from 370 ms up to 71 s.

In the case of Vbulletin Darkness instance (1010 tiles), TexturePacker and SpriteRotate were only able to give result. In fact, SpriteRotate lowers sprite size by 800 KB and transfer time by 30 s.

Through computational results, SpriteRotate was able to generate sprites to all tiles instances with up to 1010 tiles. SpriteRotate produced a transfer time of seconds compared to few tens for considered generators. This is a very substantial improvement for the objective function (1). Overall, although our solution was not designed to generate one sprite with the smallest file size, it still outperforms competitors.

In the second round of comparison, SpriteRotate has been evaluated to Spritepack. The comparison also focused on sprites size and transfer time. Due to lack of results related to Spritepack, the comparison was only performed on 5 tiles sets. The results are collected in Table 7.

For small tiles instances with up to 32 tiles, SpriteRotate was able to reduce sprites size by a factor of 1.2 to 4. In absolute terms, the reduction was from 1.5 KB to 18 KB. As a consequence, transfer time  $T(S, c)$  was reduced from 60 ms to 720 ms.



For moderate instance Oscommerce Pets (162 tiles), the improvement of transfer time, by 1.82s, was driven by reduce in sprites size by 47KB.

To conclude this experimental comparison, the proposed approach, SpriteRotate, focused on solving CSS-sprite packing using a transformation to a probabilistic non oriented bin-packing problem. The main contribution was allowing tiles rotation. SpriteRotate was compared to 9 alternative generators on tiles instances, of popular open source web applications, with up to 1010 tiles. Our experimental study has demonstrated that SpriteRotate outperformed the alternative generators.

Though SpriteRotate is not necessarily constructing optimum sprites because we are dealing with NP-Hard problem. Thus, we can conclude that tiles rotation

**Table 4.** Test instances

Instance name	Number of tiles	Tiles classification			URL
		PNG	GIF	JPEG	
Magneto Hardwood	9	3	5	1	<a href="http://www.themesbase.com/Magento-Skins/download/?dl=7396">http://www.themesbase.com/Magento-Skins/download/?dl=7396</a>
Sprite Creator	26	26	0	0	<a href="http://www.codeproject.com/KB/HTML/SpritesAndCSSCreator/SpriteCreator_v2.0.zip">http://www.codeproject.com/KB/HTML/SpritesAndCSSCreator/SpriteCreator_v2.0.zip</a>
Joomla Busines14a	29	28	0	1	<a href="http://www.joomla24.com/Joomla_2.5_%10_1_7_Templates/Joomla_2.5_%10_1_7_Templates/Business_14.html">http://www.joomla24.com/Joomla_2.5_%10_1_7_Templates/Joomla_2.5_%10_1_7_Templates/Business_14.html</a>
Mojoportel Thehobbit	32	28	3	1	<a href="https://www.mojoportel.com">https://www.mojoportel.com</a>
Squirrel Mail_outlook	73	16	57	0	<a href="https://sourceforge.net/projects/squirreloutlook/">https://sourceforge.net/projects/squirreloutlook/</a>
Myadmin Cleanstrap	198	196	2	0	<a href="https://github.com/phpmyadmin/themes/tree/master/cleanstrap/img">https://github.com/phpmyadmin/themes/tree/master/cleanstrap/img</a>
Prestashop Matrice	212	52	139	21	<a href="http://dgcrafft.free.fr/blog/index.php/category/themes-prestashop/">http://dgcrafft.free.fr/blog/index.php/category/themes-prestashop/</a>
Smf Classic	317	62	254	1	<a href="http://www.themesbase.com/SMF-Themes/7339_Classic.html">http://www.themesbase.com/SMF-Themes/7339_Classic.html</a>
Vbulletin Darkness	1010	646	351	13	<a href="https://www.bluepearl-skins.com/forums/topic/5544-darkness-free-vbulletin-skins/">https://www.bluepearl-skins.com/forums/topic/5544-darkness-free-vbulletin-skins/</a>

**Table 5.** Comparison of SpriteRotate to alternative generators on size of sprite  $f_i$  (Kb)

	Codepen	Csgencom	Cdplxsg	Stitches	Sstool	Retina	Shoebox	Txturepk	Sprite Rotate	Min	Max
Magneto Hardwood	296	738	568	23	782	831	506	746	16	5	815
Sprite Creator	113	43	437	394	473	427	453	434	15	28	457
Joomla Busines14a	33	24	15	15	23	24	15	21	5	10	28
Mojoportall Thehobbit	59	149	159	197	192	205	146	160	7	52	190
Squirrelmail Outlook	66	102	89	121	105	114	62	98	50	16	71
Oscommerce Pets	273	1601	1612	1680	1711	1903	1627	608	35	238	1868
Myadmin Cleanstrap	47	63	55	86	70	82	56	45	23	22	41
Prestashop Matrice	62	138	136	165	144	-	123	133	51	112	515
Smf Classic	107	-	220	265	239	-	133	205	25	82	240
Vbulletin Darkness	-	-	-	-	-	-	-	839	39	800	800
Average	132	357.2	365	326	415	480	346	348.35	26.96	136	502

**Table 6.** Comparison of SpriteRotate to alternative generators on objective function  $T(S, c)(s)$

	Codepen	Csgencom	Cdplxsg	Stitches	Sstool	Retina	Shoebox	TSxturepk	Sprite Rotate	Min	Max
Magneto Hardwood	11.47	28.09	21.70	12.16	30.06	31.93	19.58	28.81	0.98	10.49	38.05
Sprite Creator	4.59	1.96	16.77	15.16	18.32	16.57	17.56	16.84	0.93	1.03	17.39
Joomla Busines14a	15.92	1.25	9.15	1.11	1.22	1.27	0.92	1.17	0.55	0.37	15.37
Mojoportall Thehobbit	4.69	5.99	6.32	7.75	7.56	8.14	5.9	6.43	0.61	4.08	7.53
Squirrel Mail	2.83	4.18	3.69	4.95	4.34	4.68	2.71	4.08	2.25	0.46	2.7
Oscommerce Pets	10.6	60.53	60.94	64.12	65.37	72.66	62.17	23.45	0.73	9.87	71.93
Myadmin Cleanstrap	2.11	2.72	2.41	3.62	3.01	3.47	2.49	2.06	1.25	0.81	2.22
Prestashop Matrice	2.68	5.53	5.11	6.62	5.82	-	5.02	5.4	2.29	0.57	7.98
Smf Classic	4.37	-	8.62	10.31	9.33	-	5.40	8.14	1.3	3.07	9.16
Vbulletin Darkness	-	-	-	-	-	-	-	32.23	1.8	30.43	30.43

**Table 7.** Comparison of SpriteRotate to Spritepack on size of sprites ( $F(Kb)$ ) and objective function ( $T(S, c)(s)$ )

	Spritepack			SpriteRotate		
	$m$	$F$	$T(S, c)$	$m$	$F$	$T(S, c)$
Magneto Hardwood	3	36	1.7	1	16.7	0.98
Squirrelmail Outlook	1	8.71	0.68	1	7.31	0.62
Joomla Busines14a	1	23.76	1.25	1	5.44	0.55
Mojportal Thehobbit	7	19.31	1.08	4	7.38	0.63
Oscommerce Pets	6	84	3.54	6	36.05	1.72

have a great influence on reducing sprites size and the objective function: transfer time.

This section will conclude with some general remarks about SpriteRotate. The solution was able to provide sprites for all test sets in practically acceptable time. SpriteRotate processing time is split between image processing, geometric packing and postprocessing. The three stages consumed in average 70%, 20%, 10% of total processing time, respectively. Thus, image compression is the most time-consuming step.

Concerning image compression, we detected that for tiles with sizes lower than 1 Kb, there was not a modification in tiles sizes. As matter of fact, image compression was efficient for tiles with sizes larger than 3 Kb.

SpriteRotate is considered as a research tool and not an industrial one. In fact, image compression techniques and packing algorithms are evolving so other heuristics and image compression standards can be tried as well as integrating further input formats.

## 7 Conclusion

In this paper, we have approached the CSS-sprite packing problem into two-dimensional non-oriented probabilistic bin packing problem (2PBPP|R). We followed the relation between CSS-sprite packing and 2PBPP|R and proposed our approach which allowed for the first time to rotate tiles while generating sprites. Furthermore, in order to manage efficiently the big number of tiles, it was necessary to exploit 2PBPP heuristics. Our experiments on real-world sets validated our approach, which performs better than alternative approaches.

**Acknowledgments.** The first author extends her sincere thanks to Seifeddine Kaeuch for his help.

## References

1. Fast rollovers without preload. <http://wellstyled.com/css-nopreload-rollovers.html>. Accessed 29 September 2017
2. A thousand ways to pack the bin - a practical approach to two-dimensional rectangle bin packing. <http://clb.demon.fi/files/RectangleBinPack.pdf> Accessed 10 July 2017
3. Alakuijala, J., Vandevenne, L.: Data compression using Zopfli. Google inc. (2013). <https://github.com/google/zopfli>. Accessed 08 January 2017
4. Baker, B., Coffman, E., Rivest, R.: Orthogonal packing in two dimensions. *SIAM J. Comput.* **9**(4), 846–855 (1980)
5. Bellalouna, M.: Problèmes d'optimisation combinatoires probabilistes. Ph.D. thesis, Ecole Nationale des Ponts et Chaussées (1993)
6. Chazelle, B.: The bottom-left bin-packing heuristic: an efficient implementation. *IEEE Trans. Comput.* **32**(8), 697–707 (1983)
7. Chen, P.H., Chen, Y., Goel, M., Mang, F.: Approximation of two-dimensional rectangle packing. Technical report (1999)
8. Chen, T.C., Chang, Y.W.: Modern floorplanning based on b\*-tree and fast simulated annealing. *Trans. Comp.-Aided Des. Integr. Circ. Sys.* **25**, 637–650 (2006)
9. El Hayek, J., Moukrim, A., Nègre, S.: New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Comput. Oper. Res.* **35**(10), 3184–3201 (2008)
10. Framework, N.: Rectangle packing. <http://nuclexframework.codeplex.com/>. Accessed 25 January 2018
11. Gordon, J.: Binary tree bin packing algorithm. <https://codeincomplete.com/posts/bin-packing/>. Accessed 08 September 2017
12. Habib, A., Rahman, M.S.: Balancing decoding speed and memory usage for Huffman codes using quaternary tree. *Appl. Inform.* **4**(1), 39–55 (2017)
13. Huang, E., Korf, R.: Optimal rectangle packing: an absolute placement approach. *J. Artif. Intell. Res.* **46**, 47–87 (2013)
14. Jaillet, P.: A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Oper. Res.* **36**(6), 929–936 (1988)
15. Jaillet, P.: Analysis of probabilistic combinatorial optimization problems in euclidean spaces. *Math. Oper. Res.* **18**(1), 51–70 (1993)
16. Jeon, M., Kim, Y., Hwang, J., Lee, J., Seo, E.: Workload characterization and performance implications of large-scale blog servers. *ACM Trans. Web (TWEB)* **6**, 16 (2012)
17. Korf, R.: Optimal rectangle packing: new results. In: *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling, ICAPS 2004*, pp. 142–149 (2004)
18. Lodi, A.: Algorithms for two-dimensional bin packing and assignment problems. Ph.D. thesis, Université de bologne (1999)
19. Lodi, A., Martello, S., Vigo, D.: Recent advances on two-dimensional bin packing problems. *Discret. Appl. Math.* **123**(1–3), 379–396 (2002)
20. Marszałkowski, J., Mizgajski, J., Mokwa, D., Drozdowski, M.: Analysis and solution of CSS-sprite packing problem. *ACM Trans. Web (TWEB)* **10**(1), 283–294 (2015)
21. Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: Rectangle-packing-based module placement. In: Kuehlmann, A. (ed.) *The Best of ICCAD*, pp. 535–548. Springer, Boston (2003). [https://doi.org/10.1007/978-1-4615-0292-0\\_42](https://doi.org/10.1007/978-1-4615-0292-0_42)

22. Rehman, M., Sharif, M., Raza, M.: Image compression: a survey. *Res. J. Appl. Sci. Eng. Technol.* **7**(4), 656–672 (2014)
23. Shea, D.: CSS sprites: image slicings kiss of death. *A List Apart* (2013)
24. Stefanov, S.: Image optimization, part 3 : four steps to file size reduction. <http://yuiblog.com/blog/2008/11/14/imageopt-3/>. Accessed 29 Jan 2017
25. Taubman, D., Marcellin, M.: JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice, vol. 642. Springer Science & Business Media, Boston (2012). <https://doi.org/10.1007/978-1-4615-0799-4>
26. Velho, P., Schnorr, M., Casanova, H., Legrand, A.: On the validity of flow-level TCP network models for grid and cloud simulations. *ACM Trans. Model. Comput. Simul. (TOMACS)* **23**, 23 (2013)
27. Wium Lie, H., Bos, B.: Cascading style sheets. *World Wide Web J.* **2**, 75–123 (1997)