A challenge of packing CSS-sprites

J.Marszałkowski, J.Mizgajski, D.Mokwa, M.Drozdowski

Institute of Computing Science, Poznań University of Technology

Motivation

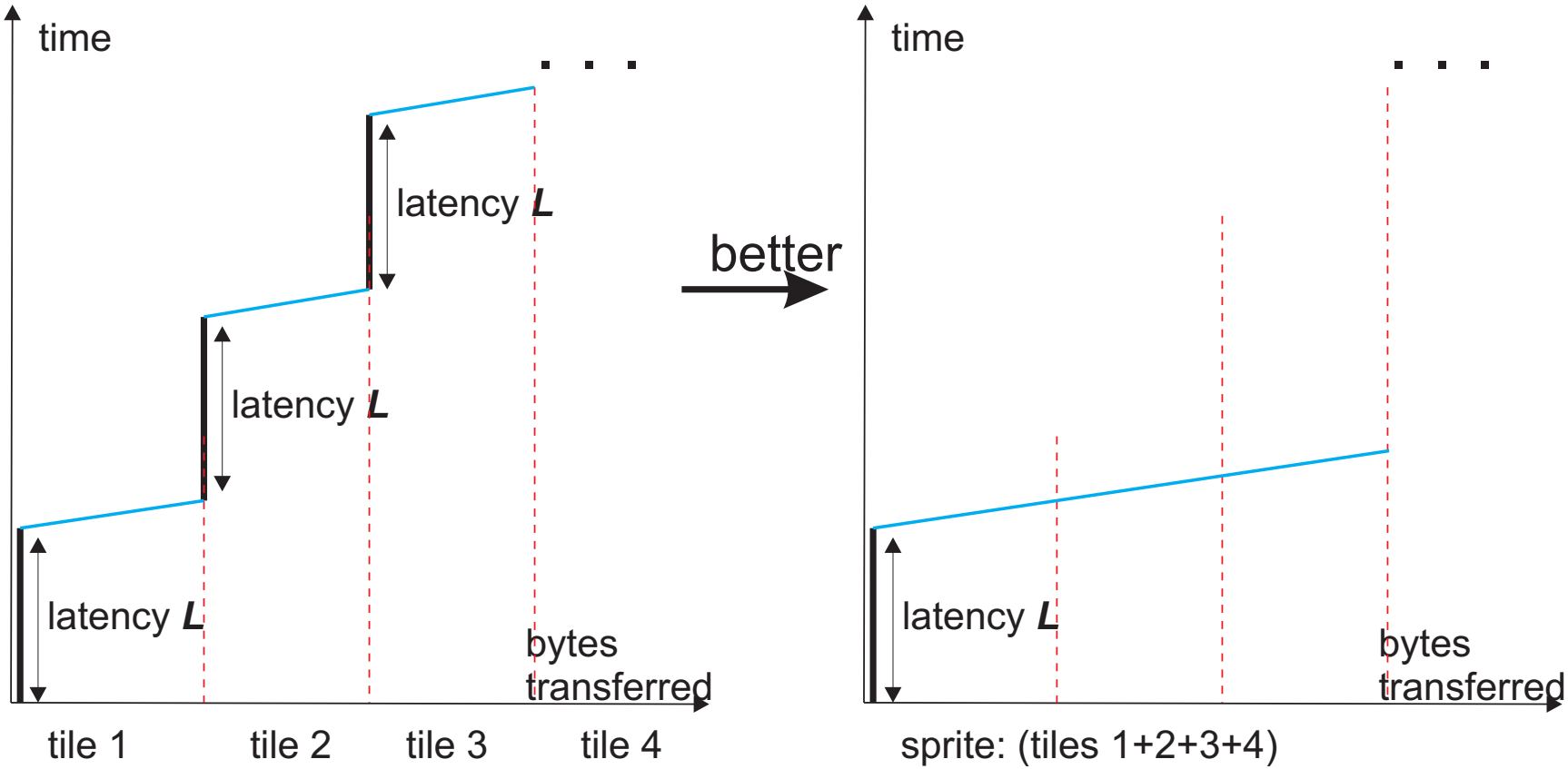
Short web page load time is important for the Internet industry.

Web pages are heavily loaded with small images (icons, buttons, backgrounds, infrastructure elements, etc.)

According to some studies over 60% of all HTTP requests for static content are images.

The interaction with a web server has a relatively long constant delay (a.k.a. latency, startup time).

Consequences of the latency (startup time)



Fetching many images separately multiplies fixed overheads and results in extensive web page loading time.

CSS-sprite packing = Spriting

CSS-sprite packing is a technique used in web design to avoid repetition of web interactions and improve performance of displaying web pages.

tiles **sprite**

Challenges

Given a set of tiles we intend to combine them into a **set of sprites** for minimum downloading time.

Factors determining downloading time:

- geometric packing,
- image compression,
- communication performance.

These three factors are tightly interrelated.

Other factors e.g. browser rendering efficiency, server cache performance, are beyond the scope of this presentation.

Geometric challenges

Tiles are rectangles, so constructing a sprite is a 2D packing/cutting problem.

HOWEVER, ...

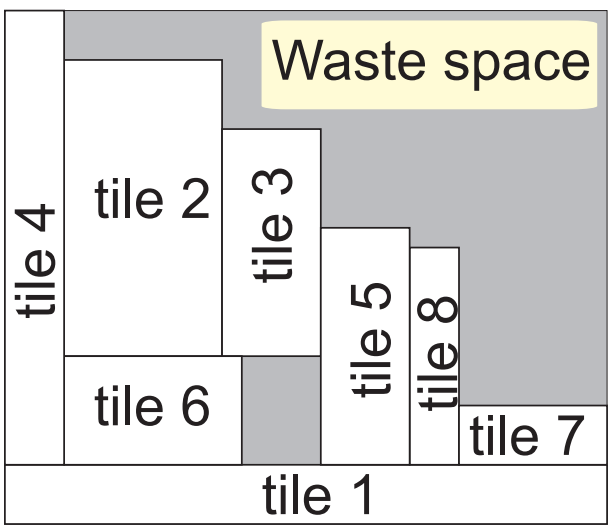
Geometric challenges, 2D cutting/packing, but...

Which packing model?

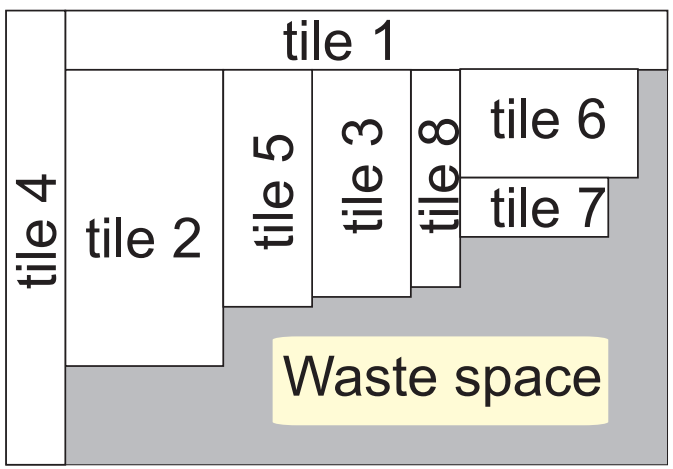
Two models of 2D cutting/packing are applicable:

- *Rectangle packing* (RP) - find the smallest area bounding box enclosing the tiles.
- *Strip packing* (SP) - put tiles on an infinite strip, there are two versions:

vertical layout



horizontal layout



Geometric challenges, 2D cutting/packing, but...

To waste or not to wase?

We do not use any material substance which 1) should be conserved, or 2) would impose a *bounding box*.

Image compression algorithms (in JPEG, GIF, PNG) are capable of dealing with the waste. So is the wasted area an issue?

$\rightarrow$ Image compression algorithms are not perfect in eliminating waste, so waste translates to communication time.

$\rightarrow$ Image area translates to **memory** usage in browser.

We **should not** waste image area!

Geometric challenges, 2D cutting/packing, but...

What is the objective function?

Communication time depends on sprite(s) sizes (in bytes).

Sprite size(s) depends not only on the sizes of tiles rectangles, **but also** on
$\rightarrow$ variety of tile compression methods (JPEG, PNG, GIF),
$\rightarrow$ tile neighborhood matters,
$\rightarrow$ complexity of the image.

Hence, the objective function cannot be calculated as easily as in the classic 2D cutting/packing.

Image compression

Image compression challenges:

- even given a solution of 2D packing, compression for minimum size is **NP**-hard,
- compression is time-consuming anyway (even ignoring combinatoric aspects)
  $\rightarrow$ only a few attempts of alternative packing and compressing are possible,
- lossy image compression (JPEG) allows for small images at the cost of quality, what level of information loss is satisfactory?
- image compressor settings $\rightarrow$ postprocessors

Communication Performance Challenges

Quality of sprites should be measured as the **downloading time**. However,

• A variety of browser, communication, server platforms, exist shared by activities with unknown timing
→ communication time is unpredictable and nondeterministic,
→ dispersion of performance parameters is unavoidable.

• Computational complexity of communication performance model
→ packet-level simulation impossible, we need a simple formula of communication time.

Communication Performance and Objective Function

Time of transferring set of sprites $\mathcal{S}$ over c concurrent channels:

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

where:
m - number of sprites,
f_i - size of spite i,
L - latency (startup time),
$B(c)$ - aggregate bandwidth over c communication channels,

And by the way, this is McNaughton's algorithm!

Communication Performance and Objective Function

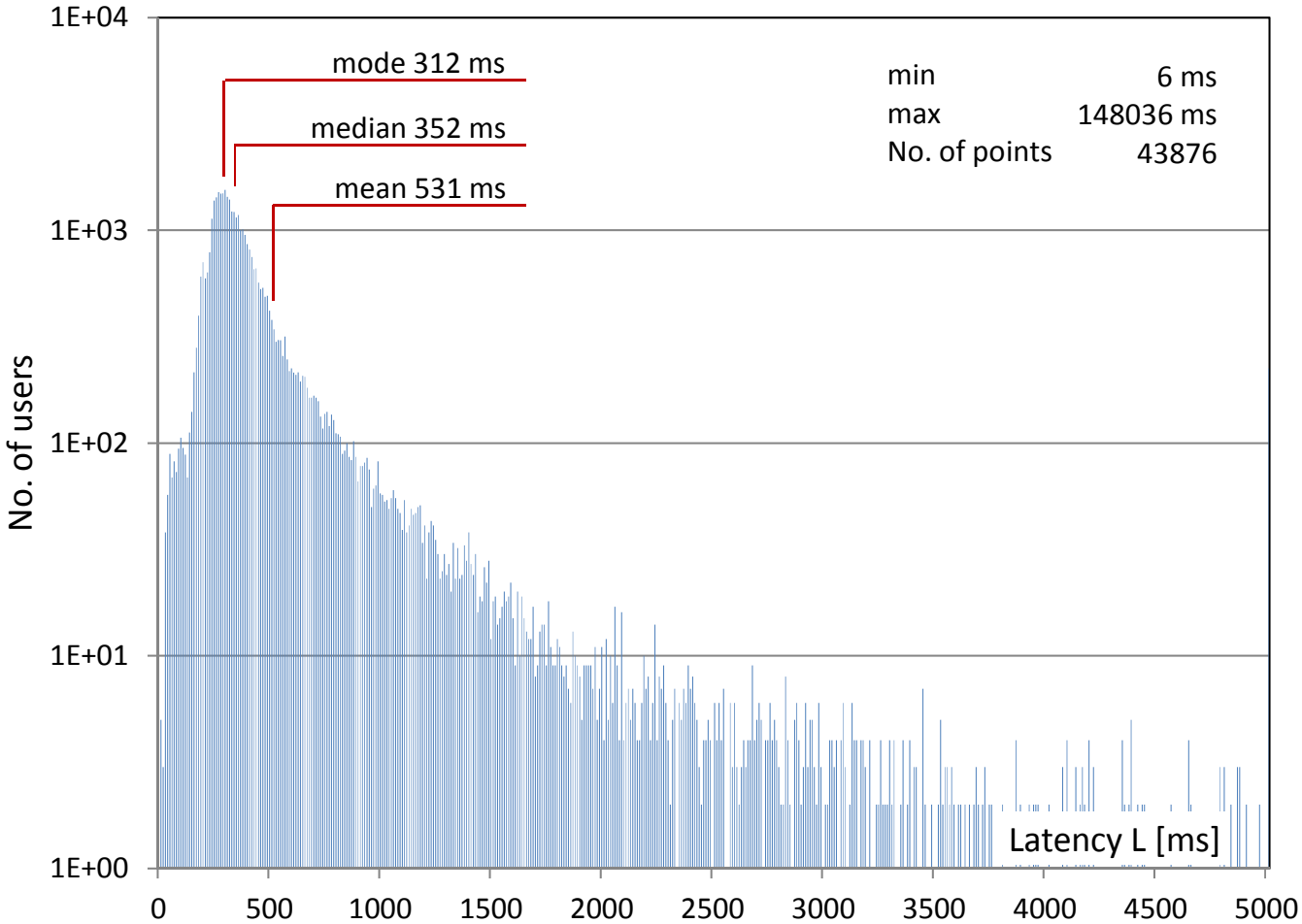
Objective function

$$T(\mathcal{S}) = \min_{c=1}^{c_{max}} \{ T(\mathcal{S}, c) \} \quad (1)$$

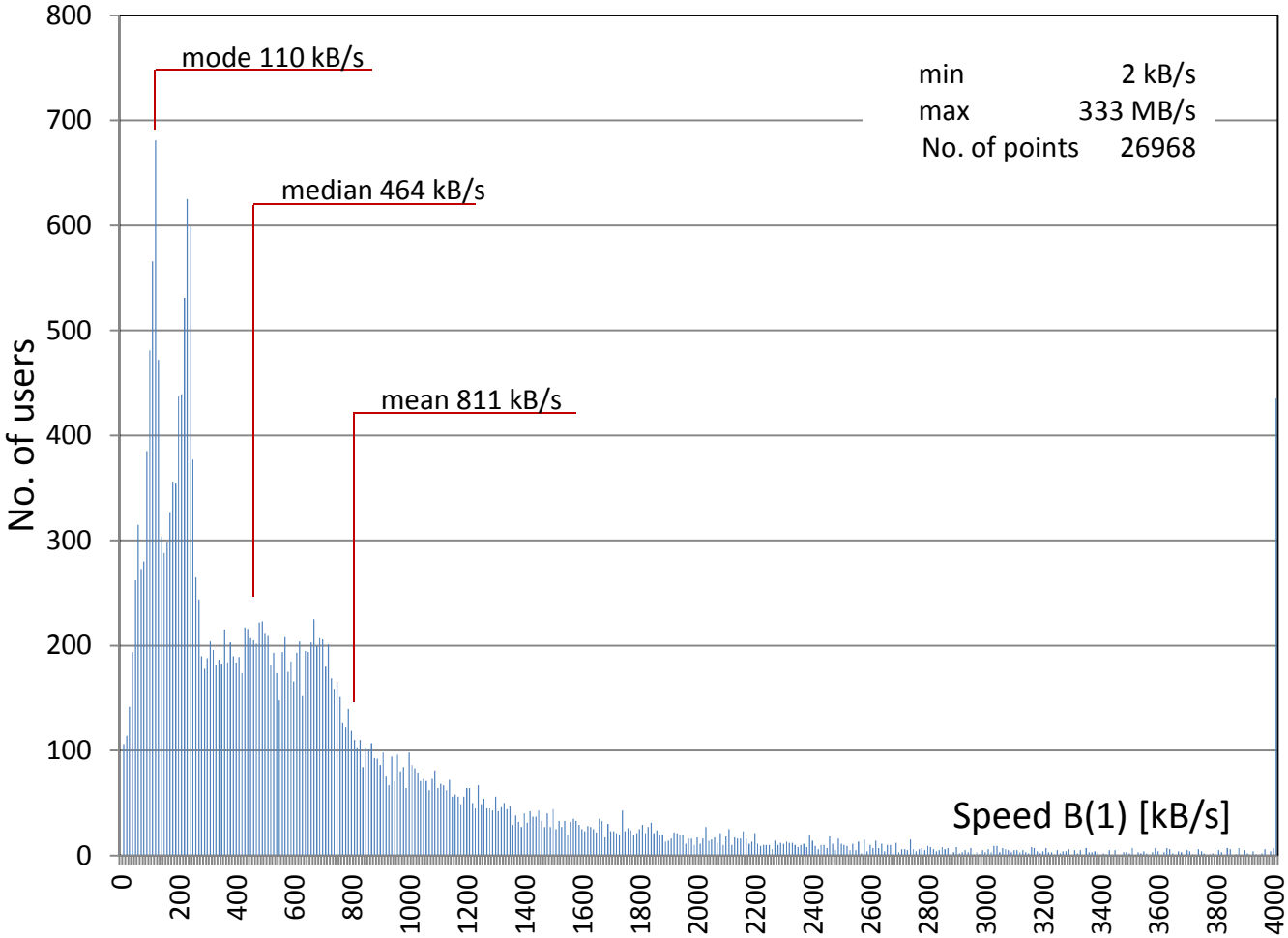
where:

c_{max} - maximum number of usable communication channels.

Latency L distribution



Bandwidth $B(1)$ distribution



Communication Speedup in Parallel Channels

Distribution of browser parallel channel number limit

Number of channels	≥ 2	≥ 3	≥ 4	≥ 5	≥ 6	≥ 7	≥ 8	≥ 9
accumulated frequency	100%	81%	68%	65%	61%	57%	12%	6%

Speedups of parallel communication

speedups	$\frac{B(3)}{B(1)}$	$\frac{B(5)}{B(1)}$	$\frac{B(7)}{B(1)}$	$\frac{B(9)}{B(1)}$
medians	1.36	1.56	1.66	1.77
SIQR	0.39	0.60	0.61	0.68

CSS-sprite Packing Problem Formulation

Given:

- set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of n tiles (JPEG, PNG, GIF images),
- communication link parameters: latency L and bandwidths vector $\overline{B}$ of length c_{max}.

Construct a set of sprites $\mathcal{S}$ such that

- objective function $T(\mathcal{S})$ defined in (1) is minimum,
- rotation of tiles is not allowed,
- each tile is comprised in only one sprite,
- each sprite is transferred in one communication channel.

CSS-sprite Packing Problem Formulation

Which means in practice:

- choose the number of sprites m,
- distribute the n tiles to m sprites,
- pack in 2D the tiles in m sprites,
- compress the m sprites,
- and do it all for minumum objective function $T(\mathcal{S})$.

Overview

Spritepack is our method to solve CSS-sprite packing problem.

Spritepack progresses in four steps:

- tile classification,
- geometric packing,
- merging with image compression,
- postprocessing.

commandline application written in C++ using MS Visual Studio 12 and Magick++ API to ImageMagick.

Tile classification

Goal:

• grouping tiles with similar sets of colors to retain as low color depth as possible.

Image classes:

- 8 bit per pixel (bpp) indexed color PNG without/with transparency (PNG8i/PNG8it),
- 8 bpp gray-scale PNG without/with transparency (PNG8g/PNG8gt),
- 24 bpp truecolor PNG without transparency (PNG24),
- 32 bpp truecolor PNG with transparency (PNG32t),
- JPEG.

All GIFs go to PNG8i, or PNG8it.
JPEG go to PNG only if it reduces size.

Geometric packing

Goals:

- grouping tiles with similar shape for packing with small waste,
- reducing runtime, a fast proxy to the full version of the algorithm.

Geometric packing **operates on rectangles** (not images) and minimizes area.

INPUT: set $\mathcal{T}$ of tiles
1: Create a group for each input tile;
2: **while** number of groups is bigger than k
2.1: $bp_1, bp_2 \leftarrow$ **nil**; $bw \leftarrow \infty$; // create an empty group pair with waste bw
2.2: **for all** unevaluated group pairs g_1, g_2 with equal image classes
2.2.1: join g_1, g_2 into a new group g_3;
2.2.2: apply to g_3 all *geometric packing strategies*;
record the packing with minimum geometric waste w_3;
2.2.3: **if** $w_3 < bw$ **then** $bp_1 \leftarrow g_1, bp_2 \leftarrow g_2, bw \leftarrow w_3$;
2.3: **endfor**;
2.4: create a new group from $bp_1 \cup bp_2$, remove bp_1, bp_2, reduce number of groups by 1;
3: **endwhile**

Geometric Packing Strategies

Strip packing:

- First-Fit,Best-Fit Decreasing Height (FFDH, BFDH),
- FFDH, BFDH with Two-Fit (FFDH2F, BFDH2F),
- Bottom-Left (BL), Modified Bottom Left (MBL).

Rectangle packing:

- Variable Height Left Top (VHLT)

k - number of output tile groups - a control parameter

The method for geometric packing is a **greedy hyperheuristic**.

Merging with image compression

INPUT: k groups of tiles

1: Create a sprite for each input tile group; record current set of sprites as solution $\mathcal{S}$ and as the best solution $\mathcal{S}^*$ with objective $T^* = \min_{c=1}^{c_{max}} T(\mathcal{S}, c)$;
2: **while** number of sprites is bigger than 1
2.1: $bs_1, bs_2, bs_3 \leftarrow$ **nil**; $bS \leftarrow \infty$; // create an empty sprite pair and empty sprite junction
2.2: **for all** unevaluated sprite pairs s_1, s_2
2.2.1: apply to the tiles in $s_1 \cup s_2$ all strategies of merging with image compression; record as s_3 the sprite with minimum size S_3;
2.2.2: **if** $S_3 < bS$ **then** $bs_1 \leftarrow s_1, bs_2 \leftarrow s_2, bs_3 \leftarrow s_3; bS \leftarrow S_3$;
2.3: **endfor**;
2.4: $\mathcal{S} \setminus \{bs_1 \cup bs_2\} \cup bs_3$; calculate objective $T = \min_{c=1}^{c_{max}} T(\mathcal{S}, c)$
2.5: **if** $T < T^*$ **then** $\mathcal{S}^* \leftarrow \mathcal{S}$; $T^* \leftarrow T$;
3: **endwhile**;

Again a greedy hyperheuristic.

This time we **opearte on images** and minimize objective function $T(\mathcal{S})$, not the area.

Strategies of Merging with Image Compression

All combinations of
geometric packing strategies $\times$ image compression methods.

Image compression methods are:
i) for PNG - minimum color depth is selected and all PNG filters are tested,
ii) if both sprites are JPEGs then JPEG formats with the baseline and progressive compression are tested (otherwise use PNG).

Postprocessing

Image sizes could have been reduced by applying different compression settings.

But it is not possible to verify alternative compression settings directly in the earlier step because it is too time-consuming.

Spritepack post-optimizes sprites obtained in the previous stage.

Applied post-processors: `pngout`, `jpegtran`.

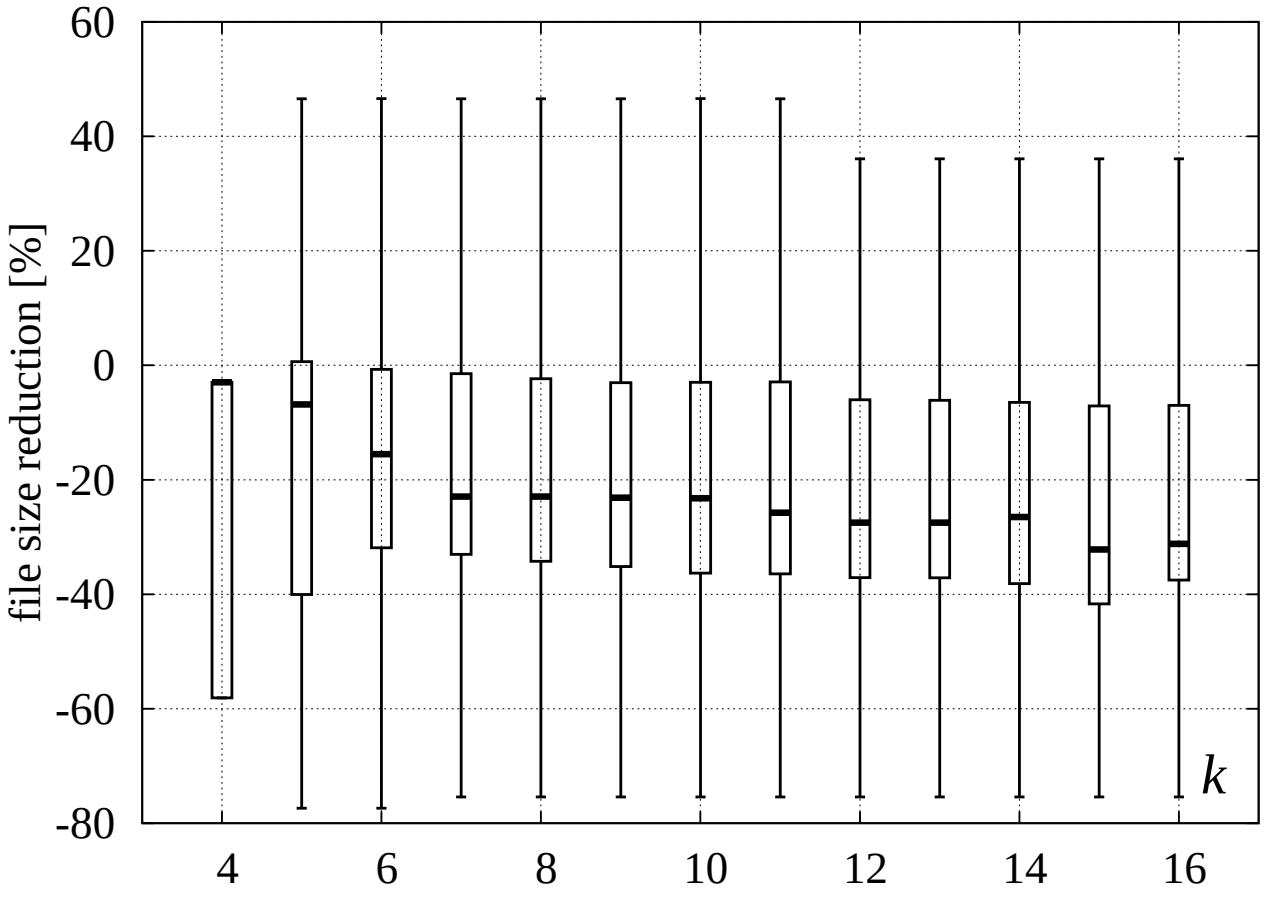
Test Setting

Instances:
32 test sets representing skins and other reusable GUI elements of popular open-source web applications ranging from $n = 5$ to $n = 1028$ tiles, 3132 images in total in GIF, PNG, JPEG.

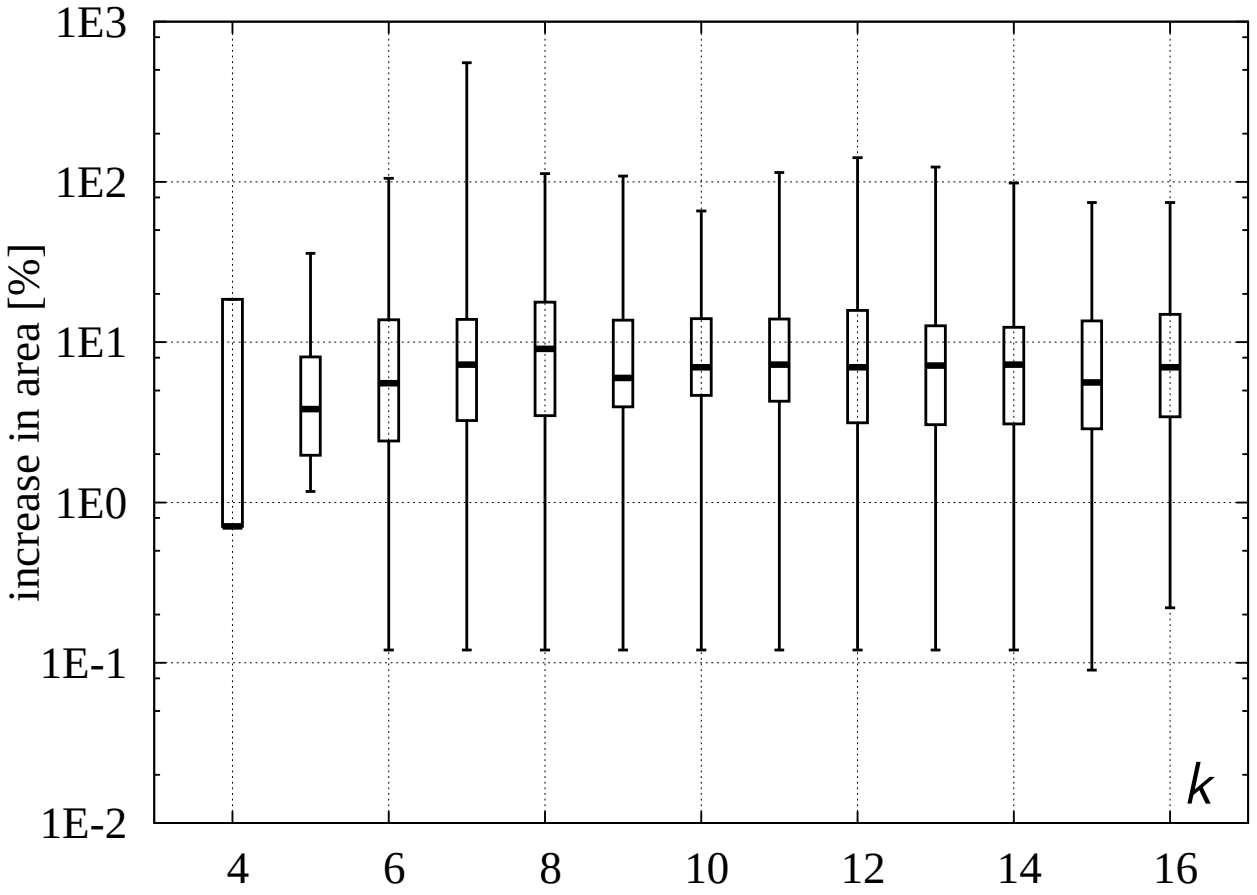
Communication parameters:
$L = 352$ms, $\overline{B} = [464, 557, 631, 685, 723, 750, 770, 791, 821]$ kB/s measured on real users.

Computing platform:
PC with i5-3450 CPU (3.10GHz), 8GB of RAM, Windows 7

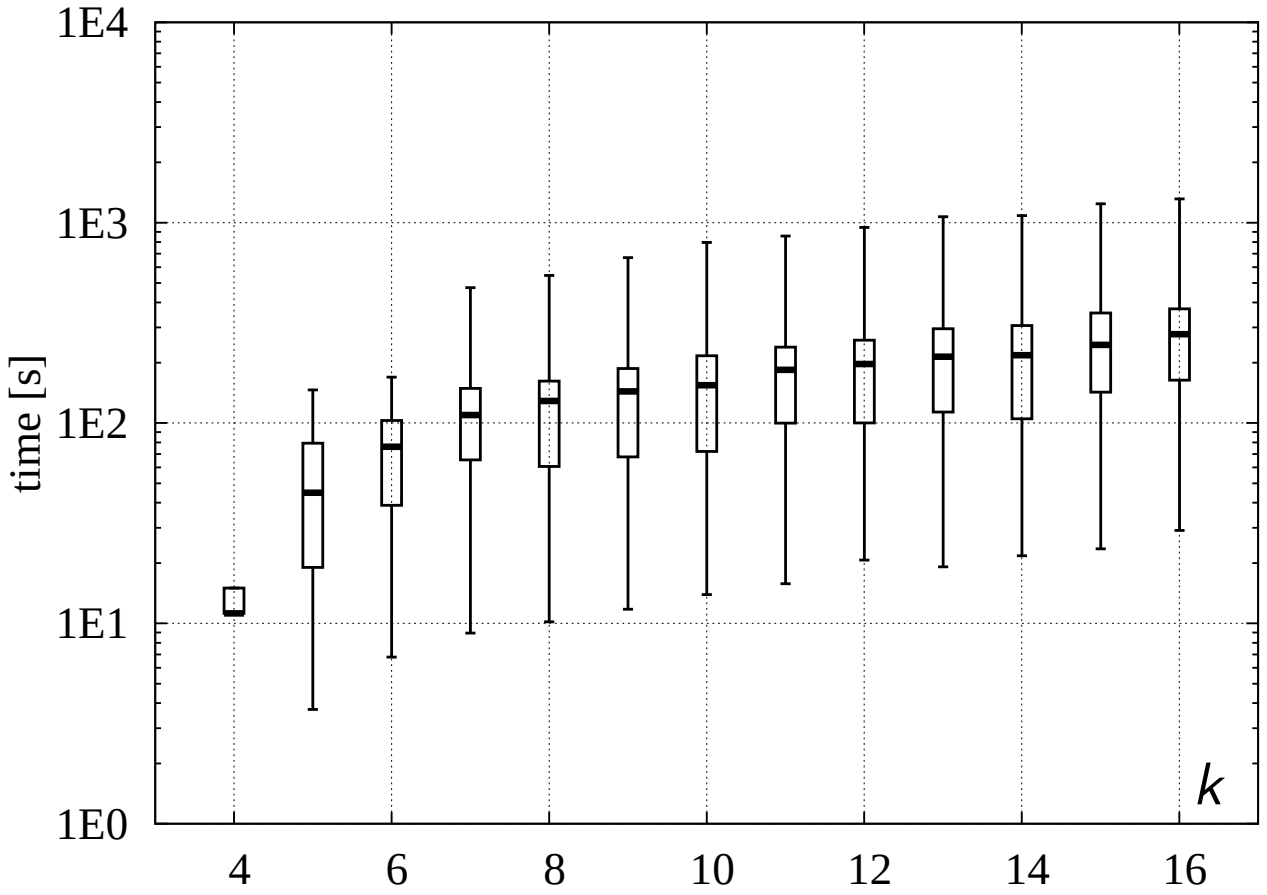
Reduction of the Total Sprite Sizes



Increase in Image Area



Runtime



The four stages consumed 5%, 1%, 81%, 13% of runtime, resp.

Comparison with other methods - file sizes

Instance name	input		shoebox		Spritepack	
	files	size [B]	sprites	size [B]	sprites	size [B]
magneto_hardwood	9	373610	1	482828	3	294128
modx_ecolife	10	50947	1	366663	3	48891
mojoportal_thehobbit	39	218993	1	726364	7	154486
oscommerce_pets	203	1201692	1	1683872	6	673785

Comparizon with other methods - download time

Instance name	medians [ms]			SIQR [ms]		
	input	shoebox	Spritepack	input	shoebox	Spritepack
magneto_hardwood	1723	764	574	1597	441	330
modx_ecolife	685	727	244	1502	427	119
mojoportal_thehobbit	776	954	302	456	539	204
oscommerce_pets	3653	1831	931	1453	872	537

Correlation between the download times and the objective function (1) was 0.952 and p-value was below 2E-06.

Conclusions

But is it still scheduling? Where is scheduling?

Scheduling is embodied in the objective function (1) modeling communication delay à la $P|pmtn|C_{max}$ solved by McNaughton's algorithm.

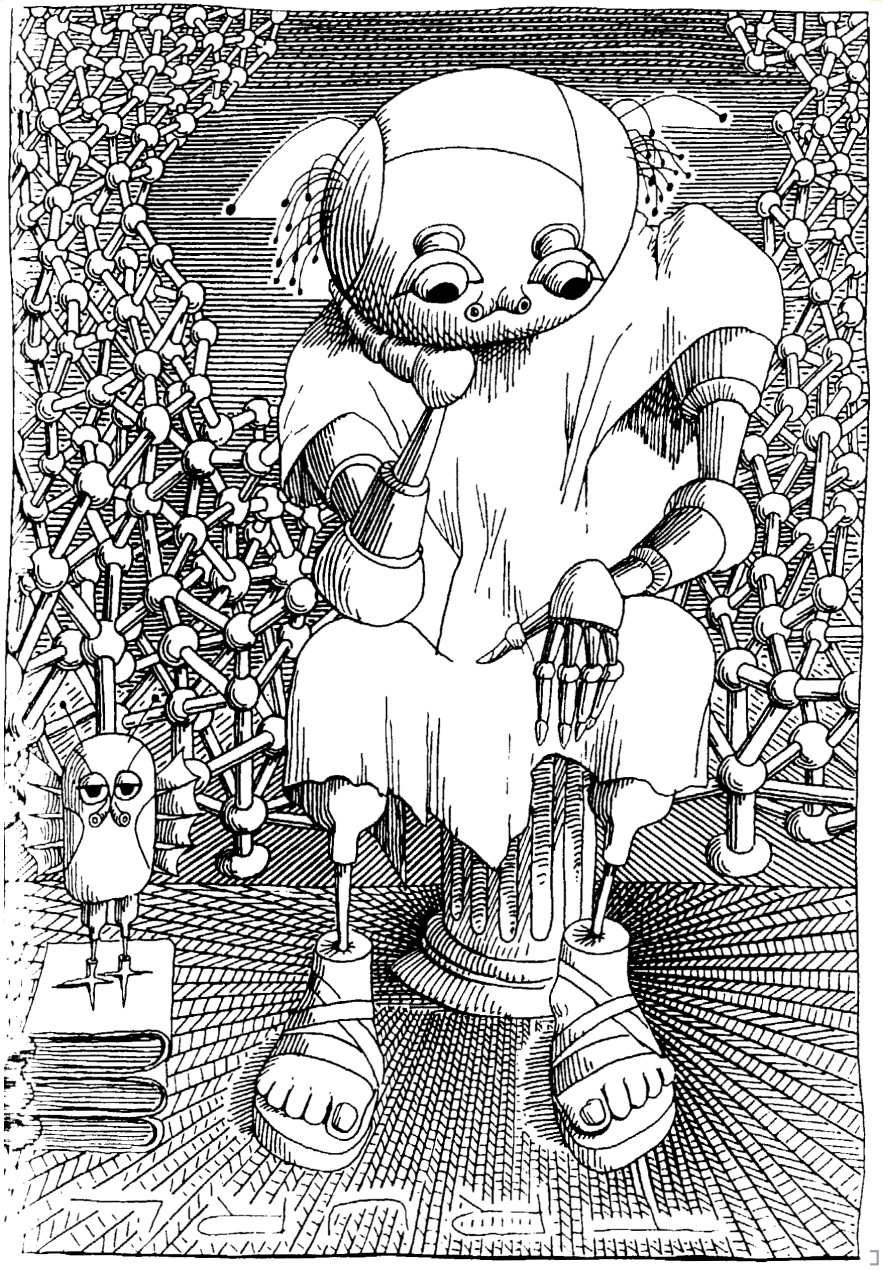
There are also other familiar toys:
- set partitioning (assigning tiles to sprites),
- 2D cutting/packing (tiles in sprites).

Toward the future worlds ...

But there is much more in this work:
- image compression efficiency,
- communication performance modeling.

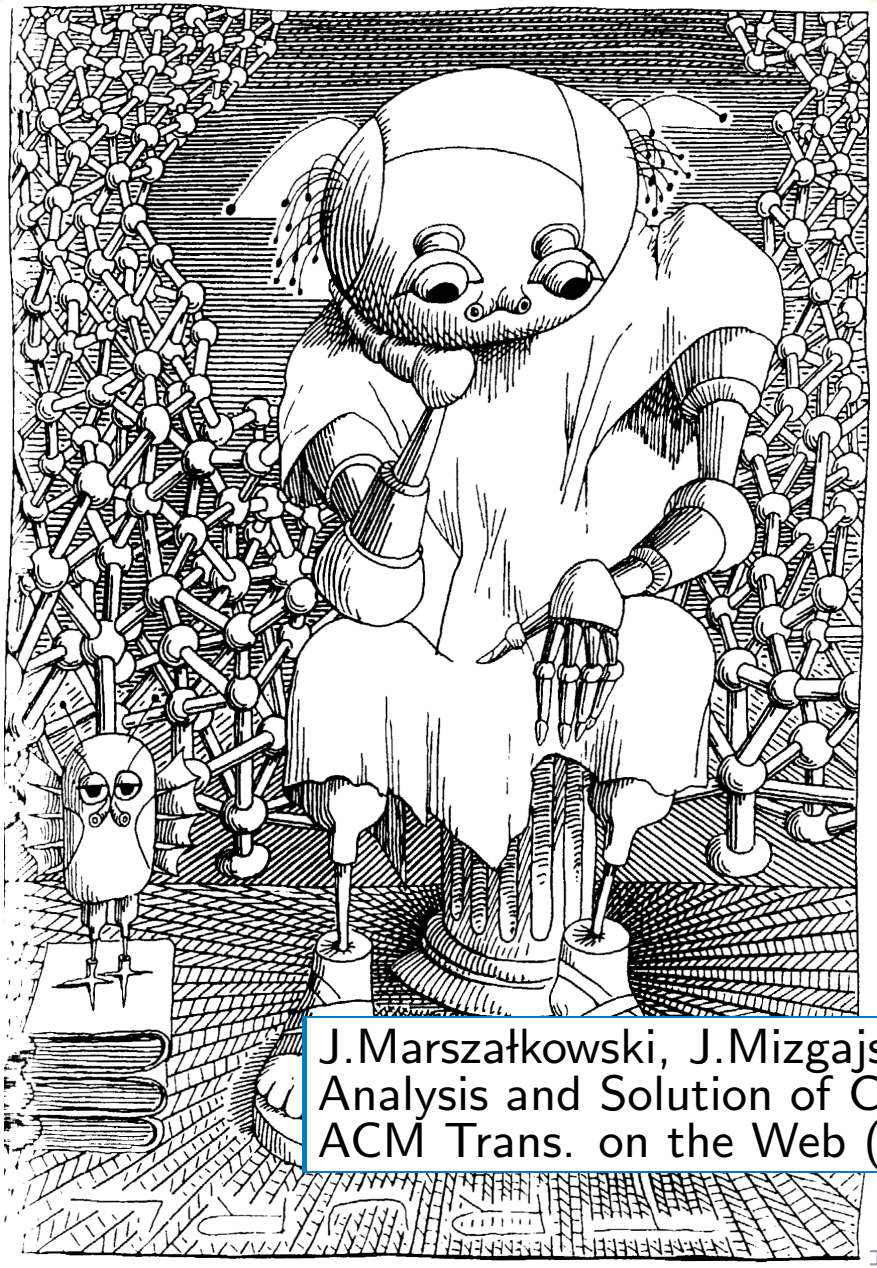
Still remain untouched:
- new/alternative technologies: HTTP 2.0, SVG, data URI.
- what do people actually fetch and see (tile filtering),
- clustering viewers according to communication performance,
- page rendering sequencing,
- in browser rendering speed and quality ...

and what have we done ...



Pic by D.Mroz, in: S.Lem, Bajki robotow. Cyberiada, Wyd.Literackie, 1965
UK: The Cyberiad – fables for the cybernetic age, 1975.

and what have we done ...



Pic by D.Mroz, in: S.Lem, Bajki robotow. Cyberiada, Wyd.Literackie, 1965
UK: The Cyberiad – fables for the cybernetic age, 1975.

J.Marszałkowski, J.Mizgajski, D.Mokwa, M.Drozdowski, Analysis and Solution of CSS-Sprite Packing Problem, ACM Trans. on the Web (10)1, Dec.2015, Art.No.1