Optimizing PDF output size of TEX documents

. . . and PDF files created by other means as well

Péter Szabó
Google

2009-09-01

EuroTeX 2009
The Hague, The Netherlands

Outline

Why create small PDF files?

- speed up downloads (and also reduce download costs)
- reduce storage costs
  - for publishers, book shops, libraries and print shops
  - save money everywhere the same PDF is stored
- use the capacity of e-book readers more effectively

To be concluded

- generate quickly, optimize later
- no dvips if possible
- find the culprit (fonts, images or drawing instructions)
- simple techniques yield the most size reduction
- optimizing drawing instructions is hard (and costs money)

Our PDF optimization approach

Steps:

1. generate the PDF as usual, adjusting only a few, crucial settings
2. repeat if necessary
3. once the final PDF is ready, optimize it automatically with one or more optimizers

Do not:

- try to improve or fine-tune every PDF creator software
- lose information (printable or interactive) while optimizing
- use a more compact output file format (such as Multiavalent compact PDF)
- render vector graphics

Proposed workflow

1. follow the best practices for choosing and configuring the TEX driver (pdfTEX, dvipdfm or ~~dvips~~ + ps2pdf)

2. if affordable, run commercial optimizer PDF Enhancer or Adobe Acrobat to optimize content streams

3. run our new optimizer called pdfsizeopt.py mainly to optimize images and Type 1 fonts
   `http://code.google.com/p/pdfsizeopt/`

4. use Multivalent `tool.pdf.Compress` to do the rest of the optimization (done automatically by pdfsizeopt.py)

Local techniques are the most effective

- remove extra whitespace and comments
- serialize strings more effectively
- compress streams with high-effort ZIP (no RLE, LZW and fax anymore)
- use cross-reference streams (with the *y* predictor)
- use object streams

Techniques if data types are known

- get rid of explicitly specified default values
- remove keys ignored by the PDF specification
- remove page thumbnails
- flatten the page structure
- inline indirect references (unless long and there are multiple referrers)

Get rid of duplicate and unused data

- get rid of unused objects (pages, images, anchors etc.)
- compact the cross-reference tables
- find duplicate or equivalent objects, and keep only one copy
- convert some inline images to objects to help deduplication
- split some large arrays and dictionaries to help deduplication

Font optimization techniques

- convert Type 1 fonts to CFF (Type 1C, Type 2)
- subset fonts
- unify subsets of the same font

Image optimization techniques

- use grayscale or a palette instead of RGB or CMYK
- use the smallest bit depth
- get rid of image duplicates based on pixel colors
- compress with multiple settings (ZIP, ZIP with predictor, JBIG2 or combinations) and pick the smallest output
- compress with high effort (e.g. slow ZIP with PNGOUT)

Advanced content stream techniques

If you can calculate on-the-paper bounding boxes, then

- get rid of objects outside the paper (and then resubset fonts)
- get rid of parts of image data outside the paper
- do not draw an object if it's covered
- clip vector graphics to the paper rectangle

Others:

- flatten form XObjects, and rebuild them if necessary
- reorganize graphic-state changing instructions
- unify small adjacent images to a large image
- separate an image for better compressibility

Drivers: dvipdfm(x) < pdfTEX ≪ dvips

- output of dvips is >50% larger than any of the other drivers
- dvips output optimized is >70% larger
- only dvips supports psfrag and pstricks
- design vector graphics with TikZ or METAPOST (with appropriate helpers) for vector graphics instead of pstricks

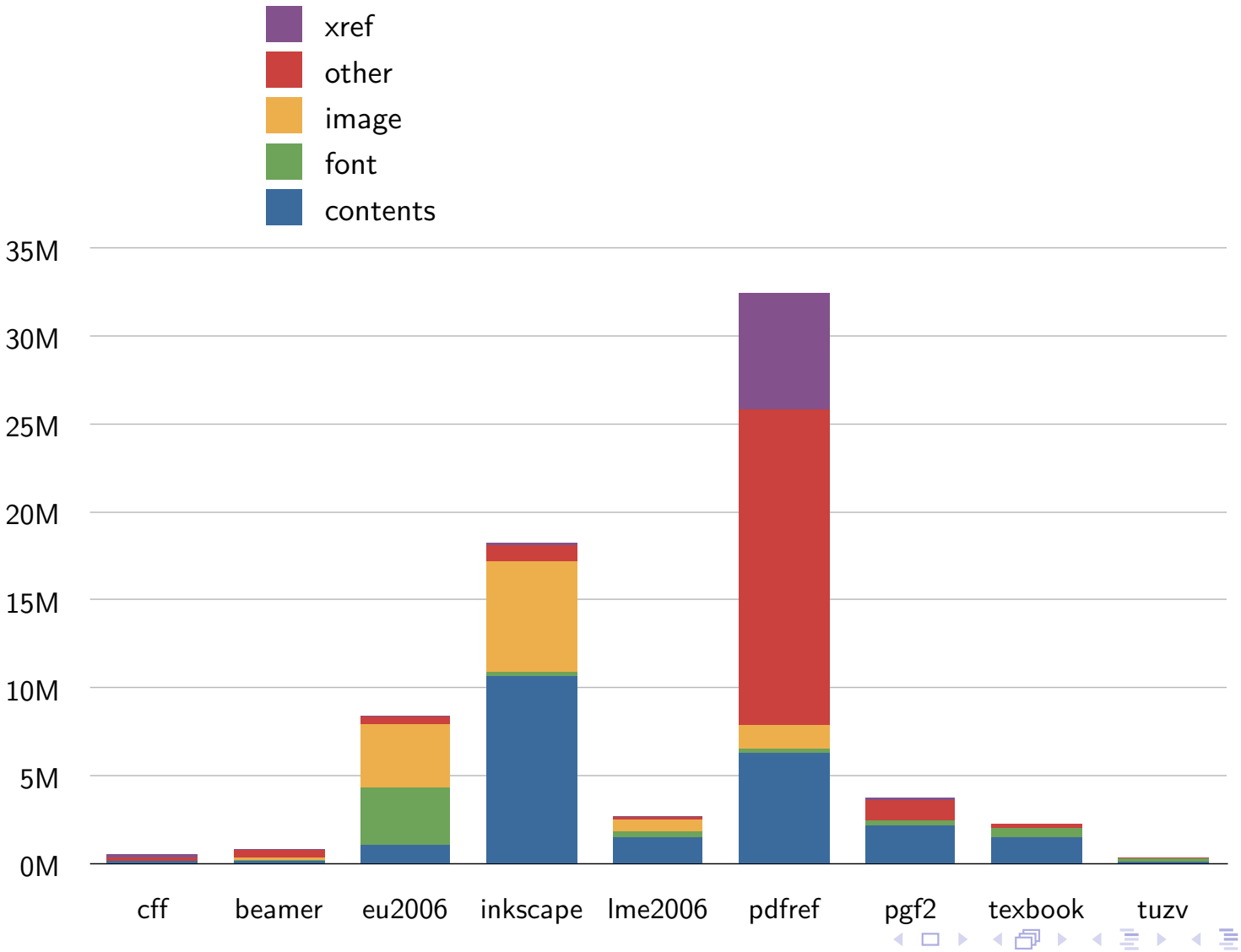
Manual setup for small PDF from TEX

- get rid of complex graphics
- reduce image resolution (300 DPI or 600 DPI): no need for a higher resolution than the printer's for the scaled image
- choose the JPEG quality
- optimize poorly exported images with sam2p
- embed vector fonts
- subset fonts (on by default for TEX text)

Input PDF files

cff	*CFF reference*; 62 pages; by FrameMaker + Distiller
beamer	first beamer.cls example; 75 slide-steps; by pdfTEX
eu2006	proceedings; 126 pages; by pdfTEX + concat
inkscape	*Inkscape manual*; 341 pages; by CodeMantra
lme2006	proceedings in Hungarian; 240 pages; by dvips + ps2pdf + concat
pdfref	*PDF 1.7 reference* 1310 pages; by FrameMaker + Distiller
pgf2	*TikZ manual* 560 pages; by pdfTEX
texbook	*The TEXbook* 494 pages; by pdfTEX
tuzv	mini novel in Hungarian; 20 pages; by dvipdfm

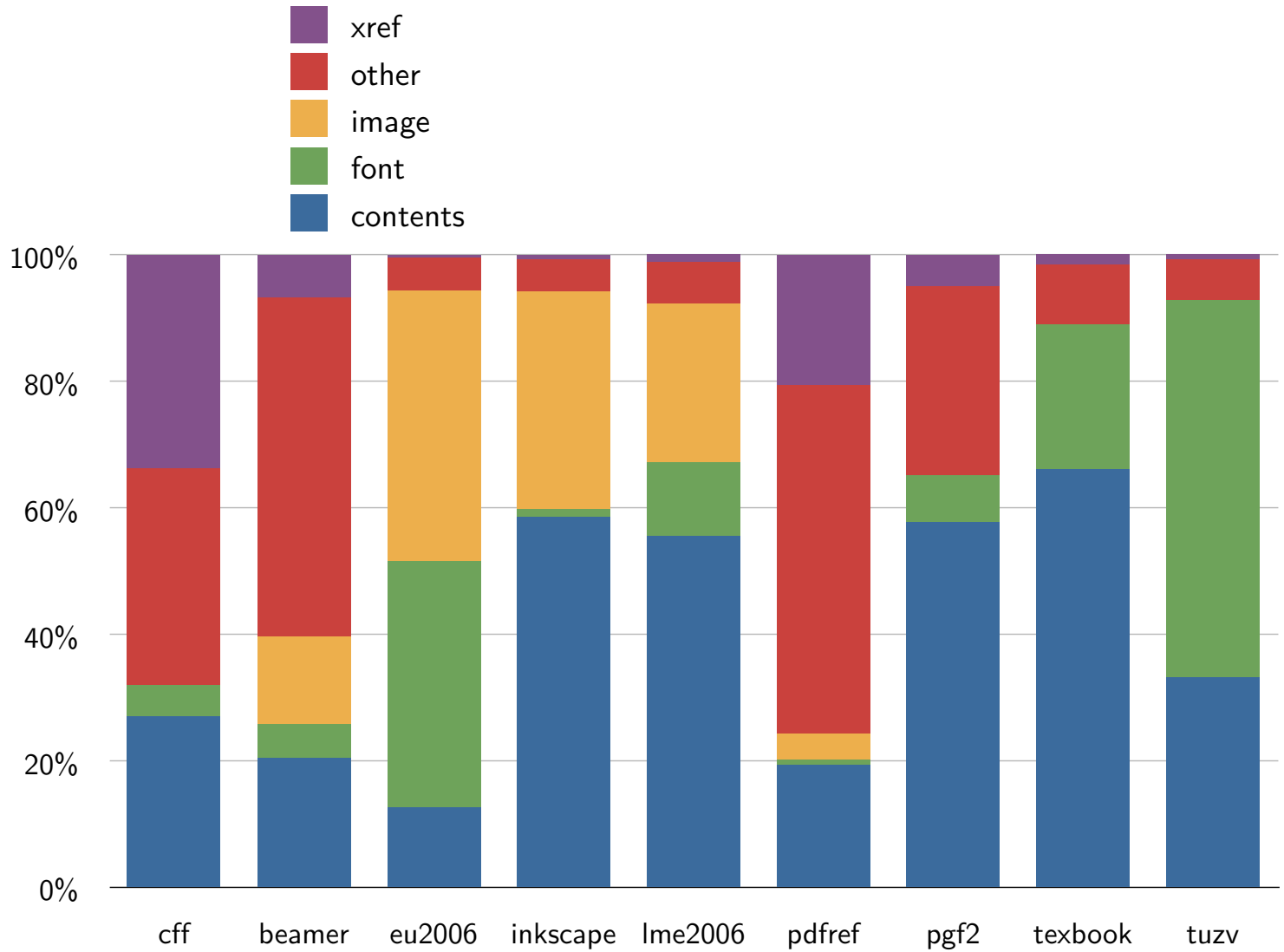
Input PDF sizes



PDF features measured

xref	cross-reference table containing the document offsets
other	hyperlinks, anchors, page structure, section structure (outlines), submittable forms, and other metadata
image	embedded pixel images (XObject and inline)
font	embedded vector font data
contents	vector graphics, text, colors, patterns etc., including content streams and form XObjects

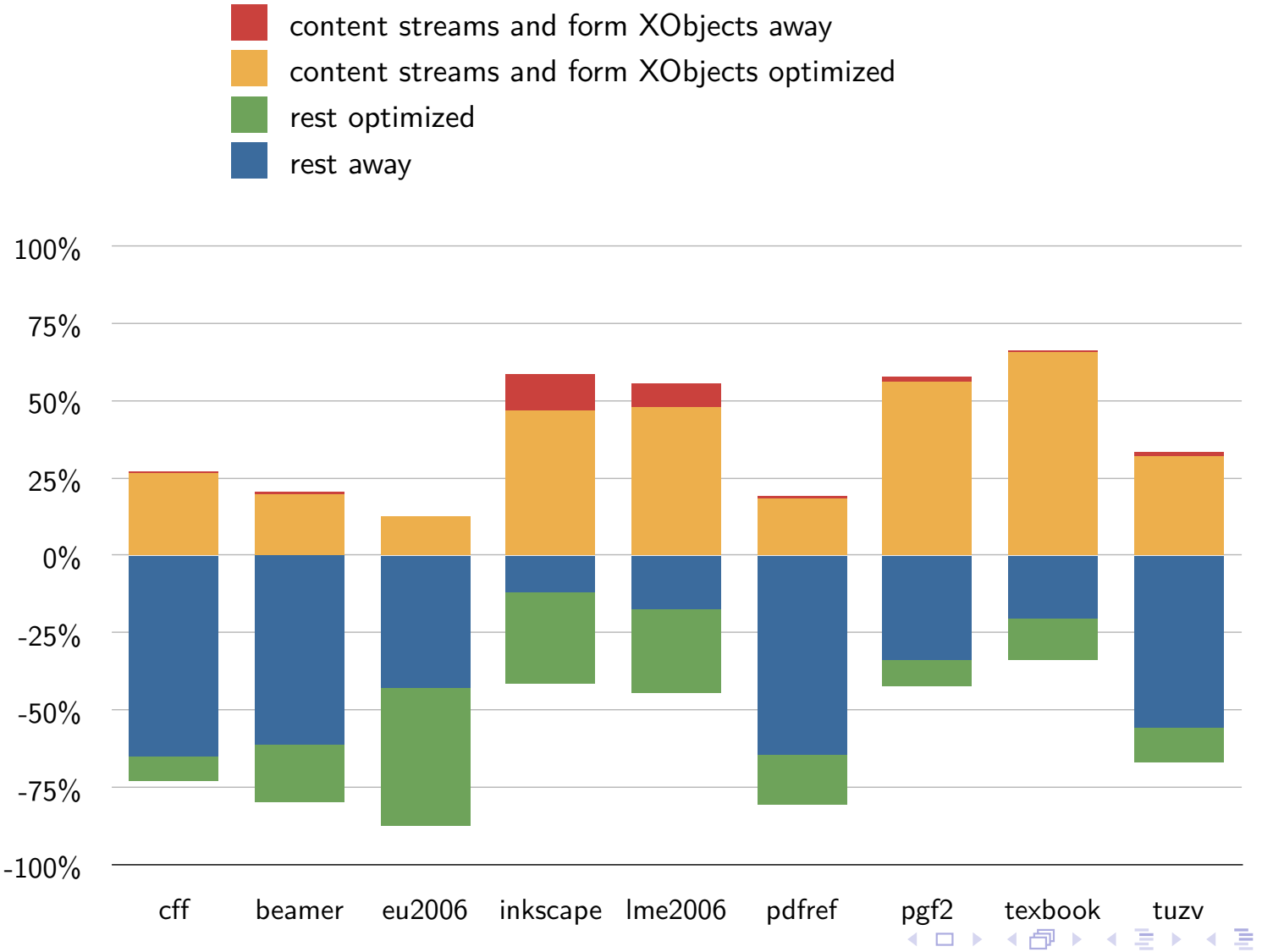
Input PDF feature distribution



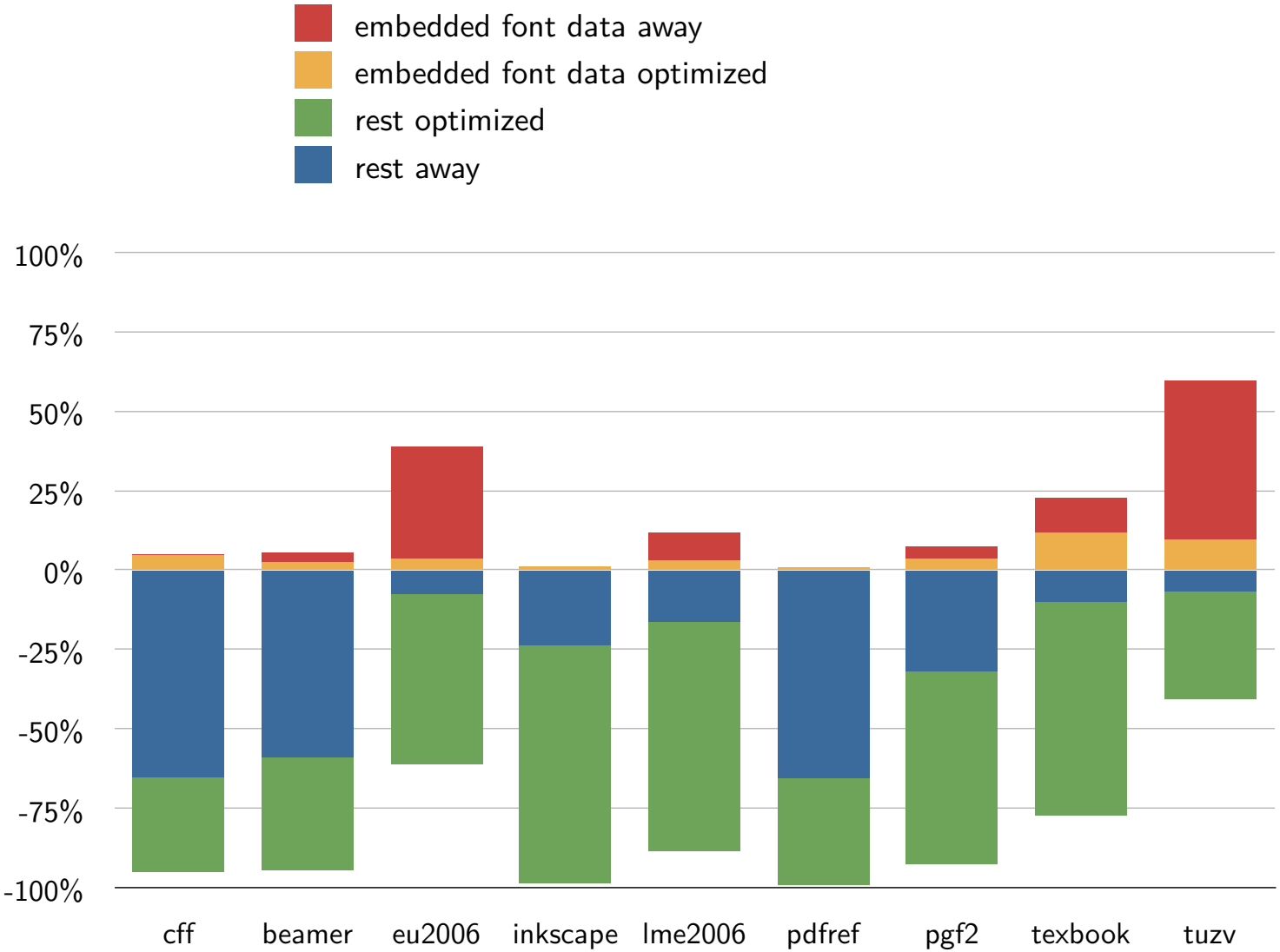
Optimizing tools measured

- input PDF files were optimized using pdfsizeopt.py (calling Multivalent in its last step)
- further reductions are possible (mostly in content streams) with Adobe Acrobat and PDF Enhancer (see in the paper)
- no information was removed or harmed

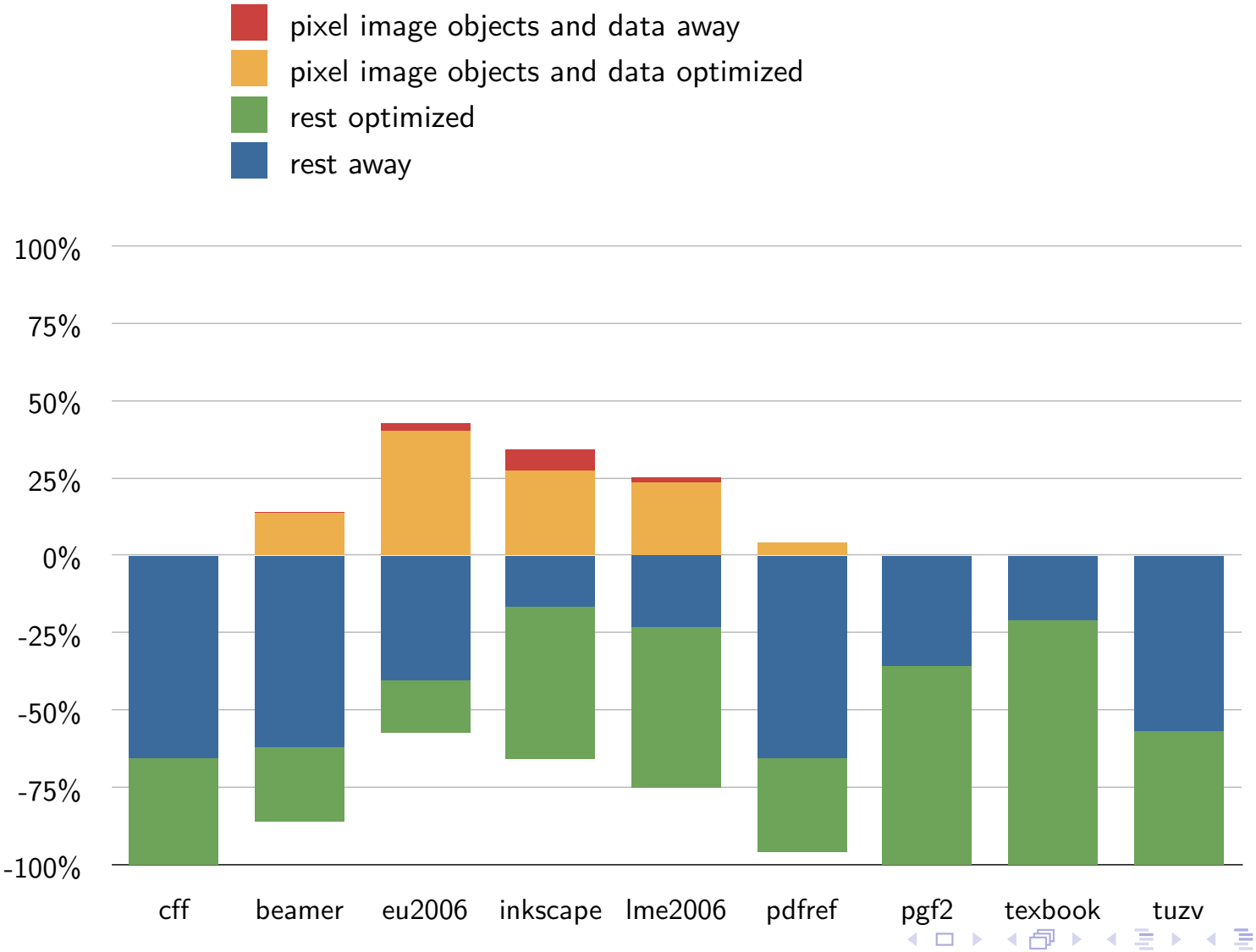
Vector graphics and text optimization effectiveness



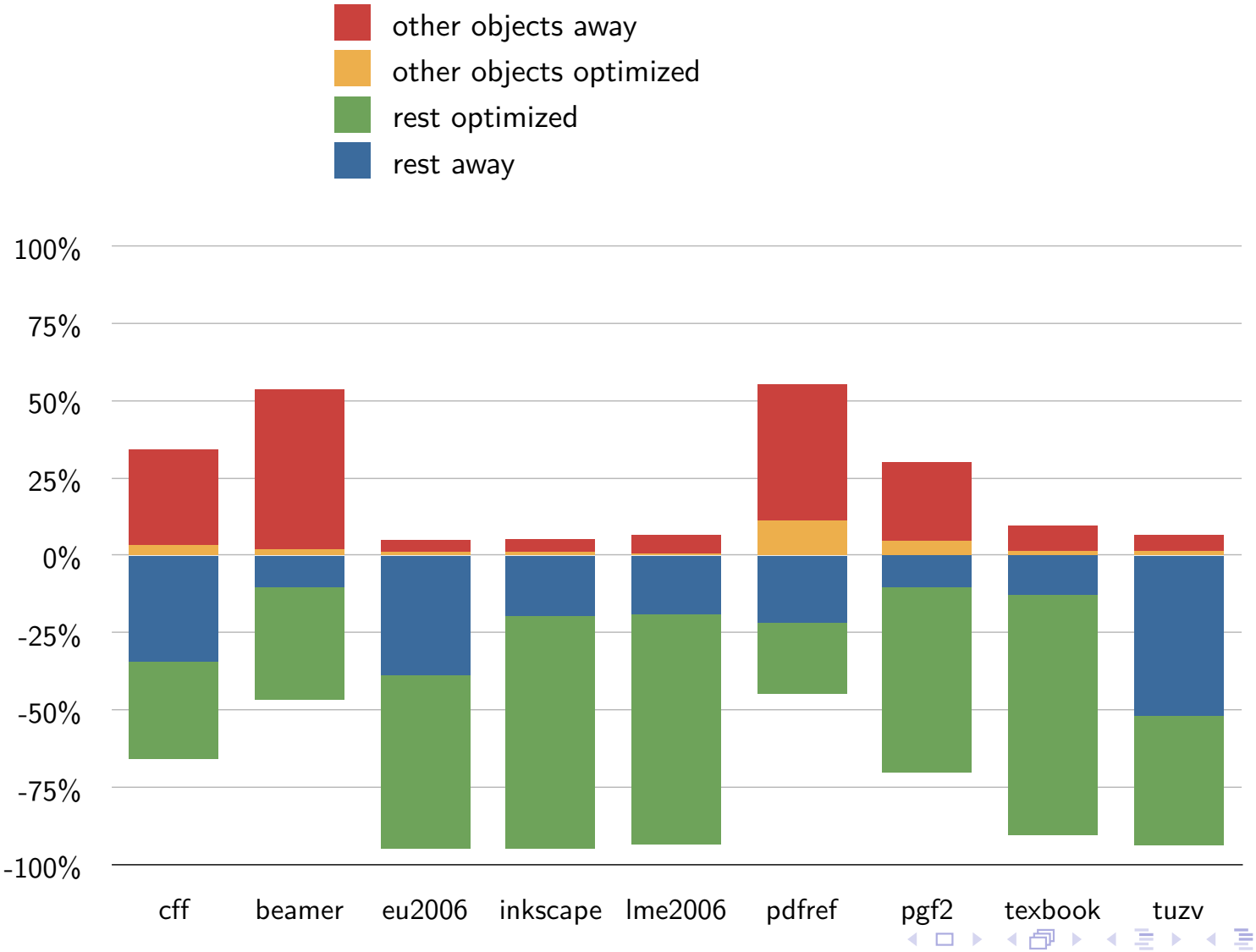
Embedded font optimization effectiveness



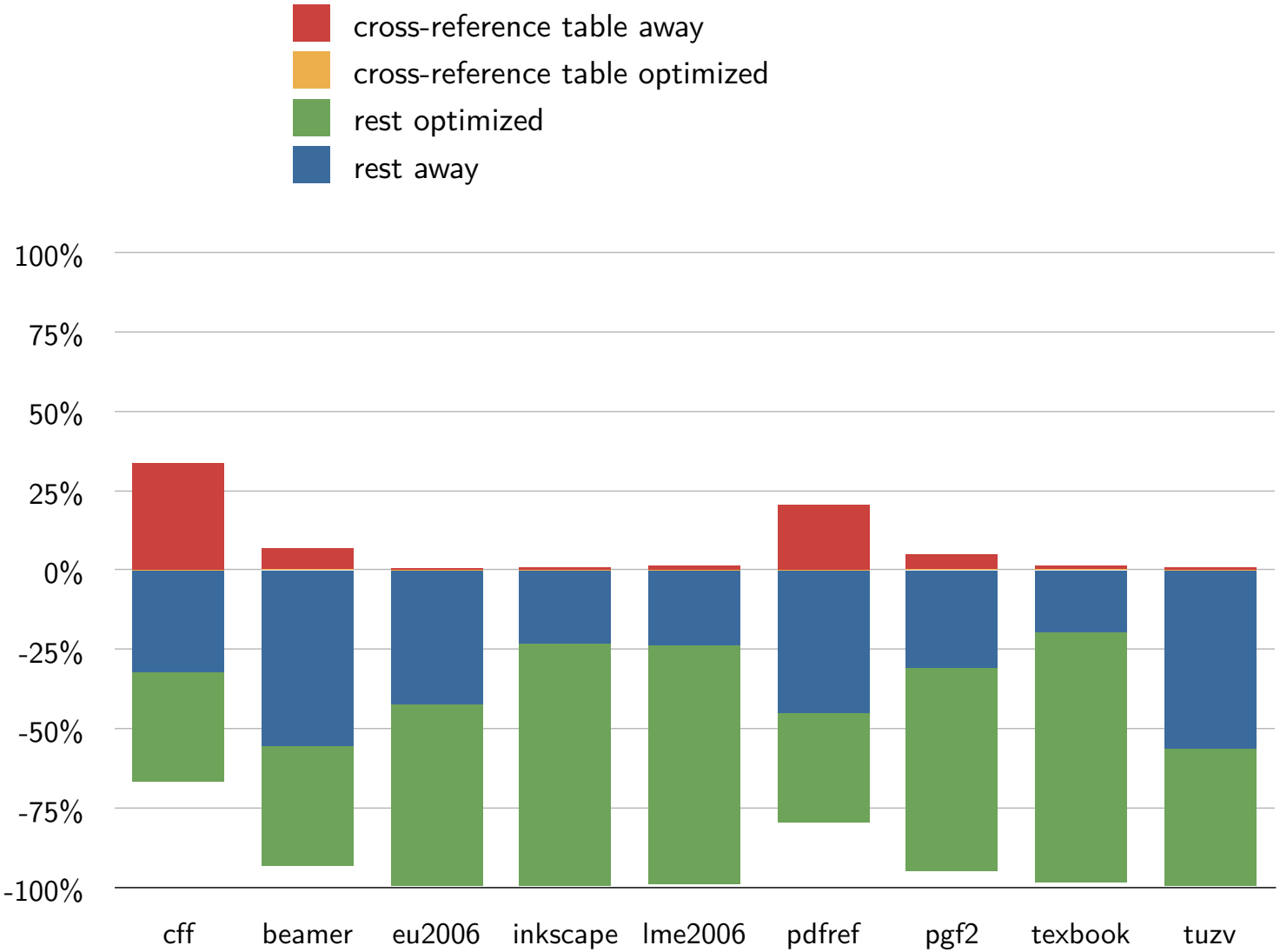
Pixel image optimization effectiveness



Other data optimization effectiveness



Cross-reference optimization effectiveness



Related work

- PDF optimization articles (mostly lossy)
- PNG optimizers
- other PDF optimizers: Multivalent, Adobe Acrobat, PDF Enhancer
- the PDF Database
- DjVu: at 600 DPI, 300% of a text-only PDF; smaller than a PDF for images
- compact PDF (30% to 60% of normal PDF)

Future work

- get rid of heavy dependencies (Python, Java, Ghostscript)
  $\longrightarrow$ C++ and Lua from the ground up
- fix shortcuts
  - support CMYK and other color spaces
  - better find mergeable CFF fonts
  - recognize all inline images
- add test PDF files (possibly from the PDF database)
- add concatenetion support for collections

Conclusion

- generate quickly, optimize later
- no dvips if possible
- find the culprit (fonts, images or drawing instructions)
- simple techniques yield the most size reduction
- optimizing drawing instructions is hard (and costs money)