http://dx.doi.org/10.15576/GLL/2020.3.31

GLL

Geomatics, Landmanagement and Landscape No. 3 • 2020, 31–41

IMPACT OF SOURCE CODE DECOMPRESSION (UNMINIFICATION PROCESS) ON THE MAP APPLICATION PERFORMANCE

Karol Król

Summary

Performance is the key usability attribute. Application performance is affected by its constituent components such as scripts, image files, and libraries. The objective of the paper is to analyse the impact of source code decompression on the performance of a map application. It focuses on two research questions. RQ1: what is the effect of source code decompression through unminification on the map application performance; and RQ2: does an application with a decompressed source code exhibit the primary (initial) performance? A prototype map application was developed to tackle the questions. Three application variants were prepared: 1) basic, 2) minified (compressed), and 3) decompressed (unminified source code). All three variants had their performance tested. The results were juxtaposed. New notions were introduced: primary performance, secondary performance, and code recycling. It was demonstrated that the secondary performance was a compromise between the primary performance and code clarity (readability).

Keywords

human-computer interaction • application design • code beautification • code recycling

1. Introduction

The quality of a website is usually assessed from the perspective of the informational value, development technique, design, and usability. A list of criteria with attributes and their values reflecting the quality of a website have been defined for each of these areas. They are the fundamental component of individual, multicriteria assessment sheets (so-called checklists) used for more or less comprehensive website audits [Dominic et al. 2011].

For many organizations, a website has become the primary communication tool, both externally and within the corporate structure. Websites are often the central platform for promotion and sale: the better the website's quality, the more effective the effort. Website quality is increasingly often associated with performance. This aspect is, in turn, regarded as being identical to the site load time. The render speed is the first thing the user notices when visiting a website, but only if it is too slow. Website perfor-

mance significantly affects target conversion such as the number of finalised transactions [Tarafdar and Zhang 2008]. Moreover, performance has a significant impact on user convenience, which made it one of the main factors in search engine rankings [Terenteva 2018].

With web services used widely in all aspects of social life, the web application performance testing is gaining wide attention [Zhu et al. 2010]. Web applications are becoming increasingly ubiquitous. Hence, the focus is on their performance. It is also the most important factor taken into account by users when evaluating systems [Wu and Wang 2010]. A particular web application design focus should, therefore, be the performance. Application performance is affected by its constituent components such as scripts, image files, and libraries [Król 2018]. Programmers often minify the source code to improve application performance. Minification is reversible if performed correctly. Code can be unminified manually or with online tools.

Universal access to new techniques and computer tools resulted in an increased focus on online geoinformation. Online maps have more and more applications ranging from citizen information to public administration. State geoinformation portals offer vast land surveying and cartographic datasets in place of analogue maps. The appeal of online maps stems mostly from the functionality of the medium itself, including the speed at which information is available [Król and Bitner 2019].

Sophisticated map portals require special software and data preprocessing. Small map components, however, can come as a compact set of files, various extensions, or hypertext documents [Król and Szomorova 2015]. The components can be developed in such a way as to make it possible to use them ad hoc, each time it is necessary to publish a thematic map online. Their usability often hinges on the performance. The objective of the paper is to analyse the impact of source code decompression on the performance of a map application. The research questions are: RQ1: what is the effect of source code decompression through unminification on the map application performance; and RQ2: does an application with a decompressed source code exhibit the primary (initial) performance?

2. Code transformation matters

There is a variety of code transformations available, ranging from the renaming of local variables to more complex code changes that affect the control flow and data flow. Transformations aimed at reducing the code size are referred to as minification. More sophisticated transformations aimed at making code comprehension and analysis more difficult are called obfuscation. When justified, such as when source file size needs to be reduced significantly or intellectual property has to be protected, minification and obfuscation can be applied simultaneously [Skolka et al. 2019].

2.1. Minification as code optimization

Minification involves the removal of all unnecessary characters from the source code while preserving its functionality. The process removes characters that improve

readability but are not needed for its execution such as new line characters, tabs, or whitespace characters. Comments, which make it easier to interpret the code (such as in HTML, CSS, or JavaScript) are removed as well [Souders 2007]. The primary goal of minification is to reduce the size of the source code to improve the performance of the program or application. Minification is a widely-accepted technique which aims at reducing the size of the code transmitted over the web [Hague et al. 2019]. It can be used to obfuscate code as well so it is less readable.

It can be difficult for a programmer to interpret an optimized source code. It is, therefore, usually one of the last steps of program development (stable release). Minification bears a resemblance to traditional code compilation. Mostly, it is applied only once right before deploying the website (therefore, its computation time does not impact the page load time) [Hague et al. 2019]. It is recommended to store the source file as it may be difficult to reverse the process or the result of the reversal can be unsatisfactory.

2.2. Code obfuscation

Code obfuscation is a transformation of the source code that preserves the semantics (application functions remain unaffected) but makes it significantly more difficult to understand the code [Ceccato et al. 2014]. JavaScript code obfuscation is a series of code transformations that turn the exposed code into a protected version of the code that is hard to understand and reverse-engineer. It should not be mistaken with minification (which optimizes the code) or encryption (as there is no password to decrypt the obfuscated code). There are tools, obfuscators, that modify the source code automatically to hinder reverse engineering.

The term 'obfuscation' is applied to a broad range of techniques, covering both syntactic and semantic transformations with results of varying efficacy. Obfuscation methods include simple renaming of variables, flattening of the control-flow graph, stripping of comments, replacement of instructions with equivalent ones, including use and emulation of entire instruction sets, and dynamic code creation [Mavrogiannopoulos et al. 2011].

There are three basic types of obfuscation transformations [Collberg 1997]: (1) Layout Transformation (layout obfuscations) involves change of identifiers, change of code formatting, or comment stripping; (2) Data Transformation (data obfuscations) involves, among other things, variable splitting, conversion of static data into procedural data, encoding modification, change of variable lifetime, or merging of scalar variables; (3) Control Transformation (control-flow obfuscations) involves such modifications as change of the flow of a loop or extension of its conditions or change of command, loop, or expression order. Some other types of obfuscation transformations such as Dead-Code Insertion, Subroutine Reordering, or Register Reassignment are used in the polymorphic and metamorphic malware [You and Yim 2010].

2.3. Unminification, reformatting, and reindenting of ugly code

Unminification is a reversal of minification. It can be useful when a developer has only a stable release with minified or obfuscated code. It can be more difficult, arduous, and time-consuming to modify such code. In principle, the effects of minification can be reversed if it involved only the removal of whitespaces and comments. It is, however, impossible in the case of obfuscated code such as JavaScript code. Only reverse engineering can help. Even if the original code structure is not restored after unminification, its readability is improved, which makes it easier to 'repair' the code.

The process of minification or obfuscation reversal is often referred to as beautification as it makes code (such as HTML, CSS, or JavaScript) 'look beautiful'. In other words, unminification makes the code readable again.

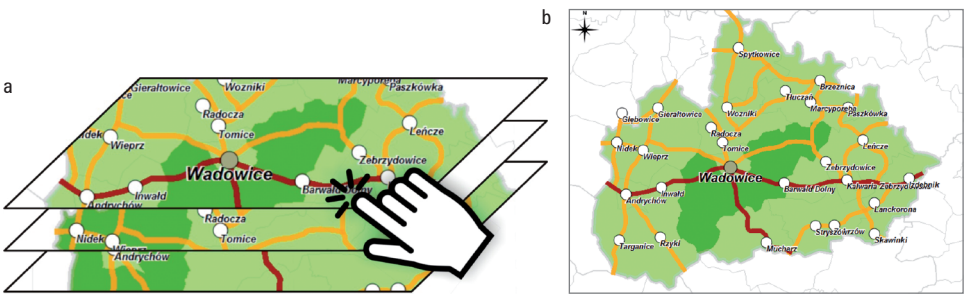
3. Materials and methods

Performance testing is normally conducted in a reasonably simulated environment with the help of performance testing tools [Subraya and Subrahmanya 2000]. The author developed a prototype web application to conduct the research. The application is made of HTML files (structure), CSS (Cascading Style Sheets, appearance), and PNG (Portable Network Graphics) image files. Its functionality is based on the jQuery JavaScript library (Table 1).

Table 1. Type and number of component files

File type	HTML	CSS	JavaScript	PNG map files	Other PNG files
Number of files	1	1	2	15	11

Source: Author's own study



Source: Author's own study

Fig. 1. The raster overlay diagram for the web application (a) and the interface of the model web application (screenshot) (b)

The application was developed to overlay rasters to be called to the browser window by the user (Fig. 1) The function of the application is to provide information. It presents contact data of municipal authorities in the Wadowicki District (Lesser Poland). The application was uploaded onto a data server.

The performance testing was conducted in three main stages [Budiman et al. 2019]: stage one S1 – preliminary testing, pre-test; stage two S2 post-optimization testing, post-test; and stage three S3 – testing following unminification, final test. The first stage involved the measurement of the initial, primary performance of the application before code minification. Next, secondary performance was tested.

3.1. Performance testing

The nature of websites poses unique software testing challenges. Webmasters, web application developers, and website quality assurance managers need tools and methods that meet their specific needs. Automated testing with special-purpose website testing software has the potential to meet these challenges [Miller 2005]. The performance testing was conducted using GTMetrix (test server region: Vancouver, Canada, Chrome Desktop) and BlazeMeter (default project, location: US East, Virginia, Google, duration: 20 minutes). Application performance was described using selected indices (Table 2).

Table 2. Online tools used in the performance tests

Performance testing tool	Website	Key performance indicators (KPIs)
GTmetrix	gtmetrix.com	1) Google PageSpeed Score 2) YSlow Yahoo Developer
BlazeMeter	blazemeter.com	3) Avg. Throughput 4) Avg. Response Time 5) Response Time 6) Avg. Bandwidth 7) Max Users

Source: Author's own study

Next, the code was minified, and the performance test was repeated (stage S2). The compression involved HTML, CSS, and JavaScript files. It was performed with selected online tools (Table 3).

The last stage (S3) was to measure the secondary performance of the application after it was unminified. Unminification was performed with selected online tools (Table 4). The resulting code was named 'recycled code' as it can be reedited.

This way, performance of three versions of the application was measured: 1) basic (initial), 2) optimized (compressed), 3) modified basic ('recycled'). The results were juxtaposed.

Table 3. Online minification tools used in the research

Compression tool	Website	Application
HTMLCompressor	htmlcompressor.com	Online HTML Compressor
CSSCompressor	csscompressor.com	Reduce CSS code size
JSCompress	jscompress.com	JavaScript minification

Source: Author's own study

Table 4. Online unminification tools used in the research

Unminification tool	Website	Application
Unminify	unminify.com	Free online tools to unminify (unpack, deobfuscate) JavaScript, CSS and HTML code, making it readable
CSSunminify	cssunminify.com	

Source: Author's own study

3.2. Key performance indicators

The GTmetrix application measures a website's performance, its load time in the browser window, and the sizes of its components. The result of the performance measurement is presented using the Google PageSpeed Score and Yahoo! YSlow indices. The YSlow attribute is expressed with a synthetic point score, ranging between 0 and 100 points [Król 2019].

The Average Throughput is the average number of Hits per Second per user during the test. This is important for understanding the amount of traffic the website or application can handle. The Average Response Time is how long, in milliseconds, it took the average user to receive a response to their request. This is important for measuring and understanding users' experience when using the site or application. A 90% Response Time is the slowest response time, in milliseconds, for the 90th percentile. This is important for measuring and understanding the majority of users' experience when using the site or application, excluding extreme incidents. The Average Bandwidth is the average amount of data transferred in KIB. This helps to understand the typical network load on the servers during high traffic scenarios [Mendelawy 2016].

4. Results

The file size of application's components was reduced as a result of minification. The compression results were a) 89% compression ratio for HTML files, b) 57% compression ratio for CSS files, and c) 40% compression ratio for JavaScript files. Note that the minification involved only the removal of whitespaces, which reduced the size of some files by half.

Unminification restores code formatting. For example, the code becomes readable, gets a hierarchical structure, and uses breake characters. In this case, the code became readable again and its formatting improved. Generally, 'recycled' files were smaller than initial files (Table 5). The size of the HTML file was slightly larger but the code was much easier to read.

Table 5. Application component file size

File type (unit)	S1 test file size	S2 test file size	S3 test file size
HTML (KB)	18.5	16.3	22.0
CSS (KB)	1.03	0.577	0.813
JavaScript (KB)	2.31	0.938	1.20
Total	21.84	17.815	24.013

Source: Author's own study

GTMetrix estimated the file size at 1.05 MB for all three variants. Neither minification nor unminification affected the application load time according to GTMatrix. Performance indices were similar or identical for all test variants. The PageSpeed Score was 70%, and the YSlow was 78%. The YSlow improved slightly only for compressed files (Table 6).

Table 6. Performance index values

Performance testing tool	Key performance indicators (KPIs)	S1 test result	S2 test result	S3 test result
GTmetrix	Total Page Size (MB)	1.05	1.05	1.05
	PageSpeed Score	C (70%)	C (70%)	C (70%)
	YSlow	C (78%)	C (79%)	C (78%)
	Fully Loaded Time	2.2 (s)	2.2 (s)	2.2 (s)
BlazeMeter	Avg. Throughput (Hits/s)	59.56	60.11	60.02
	Avg. Response Time (ms)	327.93	324.63	325.07
	Response Time (ms)	329	329	329
	Avg. Bandwidth (MiB/s)	1.09	996.68	1.3
	Max Users	20	20	20

Source: Author's own study

The Avg. Throughput (Hits/s) and Avg. Response Time (ms) were the highest for the primary performance. The secondary performance was a compromise between the primary performance and code clarity or readability (Table 6). It was the compressed version that exhibited the best performance. All the variants had the same Response Time (ms).

5. Discussion

Studies by SEMrush on 150,000 websites show that 82% of them had performance issues. Moreover, 44% of the analyzed websites had at least one critical issue (according to SEMrush Site Audit issue severity classification). At the same time, in most cases identified performance issues could be resolved with simple optimization techniques and server reconfiguration. One of the recommendations resulting from the research is to reduce website component file sizes. The heavier the page and the more resources it contains, the longer it takes to load. It is important to make sure that the page size, as well as the total size of JavaScript and CSS files (transfer size), does not exceed 2 MB. Therefore, the issue of file size remains relevant and code minification is advisable [Terenteva 2018].

Ferdinandi [2018] minified small, 25 KB JavaScript files. Minification reduced the sizes only slightly. He pointed out that the results would be of no importance for users with fast Internet connections. Still, in developing countries with serious connection speed problems such as African states where it takes more than ten minutes to upload a text file [Schuppan 2009], it may be decisive. Moreover, each reduction in file size means better application performance on mobile devices. Ferdinandi [2018] noted that minification could be very effective in the case of code with many whitespaces and long comments.

Performance plays a significant role in the success of any online venture. Pinterest increased search engine traffic and sign-ups by 15% when they reduced perceived wait times by 40% [Meder et al. 2017]. COOK's website increased conversions by 7%, decreased bounce rates by 7%, and increased pages per session by 10% when they reduced average page load time by 850 milliseconds. The BBC found they lost an additional 10% of users for every additional second their site took to load [Clark 2018]. The DoubleClick research by Google found that 53% of mobile site visits were abandoned if a page took longer than 3 seconds to load. Moreover, sites loading within 5 seconds had 70% longer sessions, 35% lower bounce rates, and 25% higher advertisement viewability than sites taking nearly four times longer at 19 seconds [Wagner 2019].

Developers typically measure a Web application's quality of service in terms of response time, throughput, and availability. Poor QoS translates into frustrated customers, which can lead to lost business opportunities [Menascé 2002]. Slow sites hurt revenue, and the opposite is also true. For Mobify, every 100-ms decrease in homepage load speed worked out to a 1.11% increase in session-based conversion, yielding an average annual revenue increase of nearly 380,000 dollars. Additionally, a 100-ms decrease in checkout page load speed amounted to a 1.55% increase in session-based conversion, which in turn yielded an average annual revenue increase of nearly 530,000 dollars.

DoubleClick demonstrated that publishers whose sites loaded within five seconds earned up to twice as much of advertisement revenue as sites loading within 19 seconds [Wagner 2019].

Obfuscation is an alternative optimization method, which can be applied to the source code. In a survey of ten top websites in the United States, minification achieved a 21% size reduction versus 25% for obfuscation. Moreover, the more JavaScript and CSS files are used and the larger they are, the higher the savings from code minification. Although obfuscation has a higher size reduction, minifying JavaScript is less risky [Yahoo! 2020].

Code minification is not everything. Eighty per cent of the end-user response time is spent on the front-end. Most of this time is tied up in downloading all the components of the page, such as style sheets, scripts, or images. Reducing the number of components reduces the number of HTTP requests required to render the page. This is the key to faster pages [Yahoo! 2020].

6. Conclusions

The paper yielded answers to the research questions: RQ1 – what is the effect of source code decompression through unminification on the map application performance? Code minification reduced the size of the component files of the application by less than 20%, which had an insignificant impact on application performance improvement (in the employed research design). Code unminification restored the performance from before compression. At the same time, it improved the code structure. RQ2 – does an application with a decompressed source code exhibit the primary (initial) performance? The application performance following decompression was similar or identical to the primary performance for most indices.

The study has demonstrated that the primary performance was the worst, although differences among variants were small. Note here that the Avg. Response Time is an inhibitor. The smaller its value the better. Neither minification nor unminification affected the Response Time.

The smaller the application component files and modifications introduced by compression, the more sensitive performance testing tool is needed. Minuscule changes in file sizes, which affect application performance, may go unnoticed by insensitive testing tools or be lost when index values are rounded off. Even the smallest changes in file size can impact application performance, especially on mobile devices.

Unminification was beneficial for the code structure and size of component files of the test application. The tests confirmed the primary advantage of unminification, which is code ordering. The unminification resulted in a tidy and ordered code with an aligned and hierarchical structure, which significantly facilitated its interpretation. Code ordering affected the application performance as well, although to a relatively small extent.

Funded with a subsidy of the Ministry of Science and Higher Education for the University of Agriculture in Kraków for 2020.

References

Budiman E., Puspitasari N., Wati M., Widians J.A. 2019. Web Performance Optimization Techniques for Biodiversity Resource Portal. IOP Journal of Physics: Conference Series, 1230 (1), 012011. https://doi.org/10.1088/1742-6596/1230/1/012011

Ceccato M., Di Penta M., Falcarin P., Ricca F., Torchiano M., Tonella P. 2014. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. Empirical Software Engineering, 19 (4), 1040–1074. https://doi.org/10.1007/s10664-013-9248-x

Clark M. 2018. How the BBC builds websites that scale. Creative Bloq. http://bit.ly/CreativeBloq [accessed: 15 May 2020].

Collberg C., Thomborson C., Low D. 1997. A taxonomy of obfuscating transformations. Department of Computer Science, The University of Auckland, New Zealand.

Dominic P.D.D., Jati H., Sellappan P., Nee G.K. 2011. A comparison of Asian e-government websites quality: using a non-parametric test. International Journal of Business Information Systems, 7 (2), 220–246.

Ferdinandi K. 2018. Does minification actually matter? Go Make Things Blog. https://gomakethings.com/does-minification-actually-matter/ [accessed: 15 May 2020].

Hague M., Lin A.W., Hong C.D. 2019. CSS Minification via Constraint Solving. ACM Transactions on Programming Languages and Systems (TOPLAS), 41 (2), 12.

Król K. 2018. Performance threshold of the interactive raster map presentation – as illustrated with the example of the jQuery Java Script component. Geographic Information Systems Conference and Exhibition „GIS Odyssey" 2018, 321–327.

Król K. 2019. Zoomlens – graphic form of data presentation on a web map, comparison of chosen tool and usage examples. Engineering for Rural Development, 18, 1641–1648.

Król K., Bitner A. 2019. Impact of raster compression on the performance of a map application. Geomatics, Landmanagement and Landscape (GLL), 3, 41–51.

Król K., Szomorova L. 2015. The possibilities of using chosen jQuery JavaScript components in creating interactive maps. Geomatics, Landmanagement and Landscape (GLL), 2, 45–54.

Mavrogiannopoulos N., Kisserli N., Preneel B. 2011. A taxonomy of self-modifying code for obfuscation. Computers & Security, 30 (8), 679–691. https://doi.org/10.1016/j.cose.2011.08.007

Meder S., Antonov V., Chang J. 2017. Driving user growth with performance improvements. Pinterest Engineering Blog. http://bit.ly/Engineering_Blog [accessed: 15 May 2020].

Menascé D.A. 2002. Load testing of web sites. IEEE Internet Computing, 6 (4), 70–74. https://doi.org/10.1109/MIC.2002.1020328

Mendelawy I. 2016. Understanding Your Reports – Part 4: How to Read Your Load Testing Reports on BlazeMeter. Broadcom. http://bit.ly/Blaze_Meter [accessed: 15 May 2020].

Miller E. 2005. Website testing. In: Companion paper of "The Web Site Quality challenge". Proceedings of QW 1998 Conference. https://bit.ly/2WIj1vp [accessed: 15 May 2020].

Schuppan T. 2009. E-Government in developing countries: Experiences from sub-Saharan Africa. Government Information Quarterly, 26 (1), 118–127. https://doi.org/10.1016/j.giq.2008.01.006

Skolka P., Staicu C.A., Pradel M. 2019. Anything to Hide? Studying Minified and Obfuscated Code in the Web. In: The World Wide Web Conference. ACM, 1735–1746.

Souders S. 2007. High Performance Web Sites: Essential Knowledge for Front-End Engineers. O'Reilly Media.

Subraya B.M., Subrahmanya S.V. 2000. Object driven performance testing of Web applications. In: Proceedings First Asia-Pacific Conference on Quality Software. IEEE, 17–26. https://doi.org/10.1109/APAQ.2000.883774

Tarafdar M., Zhang J. 2008. Determinants of reach and loyalty. A study of website performance and implications for website design. Journal of Computer Information Systems, 48 (2), 16–24.

Terenteva E. 2018. Website Performance Research. SEMrush Study 2018. http://bit.ly/35ogiZp [accessed: 15 May 2020].

Wagner J. 2019. Why Performance Matters. Web Fundamentals. Google. http://bit.ly/wagner-goog [accessed: 15 May 2020].

Wu Q., Wang Y. 2010. Performance testing and optimization of J2EE-based web applications. In: 2010 Second International Workshop on Education Technology and Computer Science. IEEE, 2, 681–683. https://doi.org/10.1109/ETCS.2010.583

Yahoo! 2020. Best Practices for Speeding Up Your Web Site. https://yhoo.it/37ECOi9 [accessed: 15 May 2020].

You I., Yim K. 2010. Malware obfuscation techniques: A brief survey. In: International Conference on Broadband, Wireless Computing, Communication and Applications. IEEE, 297–300.

Zhu K., Fu J., Li Y. 2010. Research the performance testing and performance improvement strategy in web application. In: 2010 2nd International Conference on Education Technology and Computer. IEEE, 2, 2–328. https://doi.org/10.1109/ICETC.2010.5529374

Dr Karol Król
Uniwersytet Rolniczy w Krakowie
Katedra Gospodarki Przestrzennej i Architektury Krajobrazu
al. Mickiewicza 24/28, 30-059 Kraków
ORCID: https://orcid.org/0000-0003-0534-8471
e-mail: k.krol@onet.com.pl
http://digitalheritage.pl