Exploring Genetic Improvement of the Carbon Footprint of Web Pages

Exploring Genetic Improvement of the Carbon Footprint of Web Pages

Haozhou Lyu[1], Gregory Gay[1][0000−0001−6794−9585], and Maiko Sakamoto[2]

[1] Chalmers | University of Gothenburg, Gothenburg, Sweden
haozhou@student.chalmers.se, greg@greggay.com
[2] University of Tokyo, Tokyo, Japan
m-sakamoto@k.u-tokyo.ac.jp

Abstract. In this study, we explore automated reduction of the carbon footprint of web pages through genetic improvement, a process that produces alternative versions of a program by applying program transformations intended to optimize qualities of interest. We introduce a prototype tool that imposes transformations to HTML, CSS, and JavaScript code, as well as image resources, that minimize the quantity of data transferred and memory usage while also minimizing impact to the user experience (measured through loading time and number of changes imposed).
In an evaluation, our tool outperforms two baselines—the original page and randomized changes—in the average case on all projects for data transfer quantity, and 80% of projects for memory usage and load time, often with large effect size. Our results illustrate the applicability of genetic improvement to reduce the carbon footprint of web components, and offer lessons that can benefit the design of future tools.

Keywords: Carbon Footprint · Energy Consumption · Web Development · Genetic Improvement · Genetic Programming

1 Introduction

Climate change, caused by increasing concentrations of greenhouse gases in the atmosphere, is expected to have profound long-term consequences to the human health, safety, and quality of life. The carbon dioxide emitted through development and use of software is a major contributor to climate change [2].

In this study, we focus on the carbon footprint of web pages. Web pages are some of the most commonly used programs in the world—presently, there are estimated to be approximately two billion websites, with 400 million actively maintained, and almost four billion internet users around the world [10]. Therefore, reductions in the carbon footprint of web pages could contribute significantly to reducing the overall carbon footprint of the software industry.

Reducing software carbon footprint is not a straight-forward task. Researchers have begun to make recommendations (e.g., [14, 22, 24]). Such guidelines are highly important, but can be difficult to apply—especially after the code has been written. Therefore, we are interested in exploring *automated* reduction of carbon footprint through transformation of existing source code—e.g., reducing energy consumption while maintaining the semantics of the original code.

A promising technique to impose such changes is *genetic improvement* [3], where alternative versions of a program are scored according to qualities of interest—called fitness functions—then evolved over many generations to optimize these scores. Genetic improvement has been applied to energy consumption (e.g., [3, 9, 14, 15, 23]), but not to carbon footprint or web pages.

In this study, we have explored the use of genetic improvement to automatically reduce the carbon footprint of web pages through the development of an extendable tool. This tool imposes transformations to HTML, CSS, and JavaScript code, as well as image resources, to identify web page modifications that minimize fitness functions that correlate with carbon footprint—the quantity of data transferred and memory usage—and functions intended to preserve the user experience—the page load time and number of changes imposed.

In an experimental evaluation on 10 open-source projects of varying complexity, we compare our tool to two baselines—the original, unmodified page and randomized changes. In the average case, our tool was able to reduce the quantity of data transferred for all projects, and the memory consumption and loading time for eight of the projects. Our tool outperformed the random baseline with large effect size in 92.50% of comparisons. Compressing and converting images and removing unused CSS were the most common actions performed by the tool, with major effects on solution fitness.

Our results illustrate the applicability of genetic improvement to reduce the carbon footprint of web components, and offer lessons that can benefit the design of future tools for this purpose. We make our tool[3], comparison baselines[4], and experiment data[5] available to researchers and practitioners to use or extend.

2 Background and Related Work

Carbon Footprint and Energy Consumption of Software: The carbon footprint of a software product is the quantity of carbon dioxide emissions associated with its development. There are sources of emission at multiple points, including implementation, testing, delivery, usage, and maintenance [21]. In this research, our scope is restricted to emissions associated with energy consumption during *usage*—i.e., when interactions take place with the software.

Carbon footprint is affected by the *quantity* of energy consumed and *where and how* that energy is produced or consumed, as some energy sources have a greater carbon footprint than others. Software carbon footprint is potentially affected by energy usage on both the client-side (i.e., consumer devices) and server-side (i.e., data centers in disparate geographic areas), as well as by network transmissions between the two [1]. In this study, we focus on energy usage.

Genetic Improvement: Genetic improvement is the automated improvement of non-functional qualities (e.g., performance) of software through transformations to the source code [3].Transformations are imposed using genetic programming [3], a process where population of patches are produced, their quality is

[3] https://github.com/haozhoulyu416/ARCFW-Tool
[4] https://github.com/haozhoulyu416/ARCFW-Random-Solution-Generation
[5] https://doi.org/10.5281/zenodo.8347915

measured according to one or more "fitness functions" related to the qualities of interest, and then patches are evolved over a series of generations. The patches that yield the best scores in each generation form a new population through mutation—where stochastic changes are introduced—or crossover—where aspects of "parent" patches combine to form new "children". Carbon footprint can be considered a quality. Thus, genetic programming could be used to automate its reduction through appropriate transformation actions and fitness functions.

Related Work: Past research has offered guidelines on how to reduce energy consumption and carbon footprint (e.g., algorithm selection [14], code structure [24], considering server distribution and location [14], or controlling image quality [22]). Such guidelines are highly important, but are not always easy to apply. Nor is it simple to manually improve code after it has been written. Therefore, we are interested in automated carbon footprint reduction techniques.

We are unaware of automated tools targeting carbon footprint. However, there have been several approaches targeting energy consumption. Much of this research utilizes genetic improvement—for example, Bruce et al. targeted C implementations of MiniSAT [3], Dorn et al. explore trade-offs between energy consumption and output fidelity [9], Manotas et al. optimize applications that use Java collections [14], Mrazek et al. targeted microcontrollers [15], and White et al. targeted pseudorandom number generators [23]. Other approaches include, e.g., specialized compilers [20] and data migration strategies [6]. None of these approaches target web pages, despite their prevalence, and their approaches are not applicable to this domain. Our approach is the first to target web page carbon footprint using genetic improvement.

3 Methodology

Our study is guided by the following research questions:

- **RQ1:** What factors of web page design suggest code transformations or fitness functions for genetic improvement of carbon footprint?
- **RQ2:** What impact does our prototype genetic improvement tool have on the carbon footprint of web pages?
- **RQ3:** Which program modifications are used most often by the tool?

To answer these questions, we applied the Design Science methodology to develop a prototype automated carbon footprint reduction tool for web pages. Design science is a systematic, cyclic methodology for the creation and evaluation of artifacts. This prototype tool was designed over three iterations. In Iteration I (Section 3.1), we identified factors of web page design that have a potential impact on carbon footprint. These factors form the basis of fitness functions and transformation actions in our tool. In Iteration II (Section 3.2), genetic programming—based on the NSGA-II algorithm—was utilized to develop the automated carbon footprint reduction tool. In Iteration III (Section 3.3), we conducted parameter tuning experiments, then we conducted final experiments comparing the tool against two baselines—the original web page and random solution generation.

3.1 Iteration I (Scoping, Fitness Functions, and Transformations)

The purposes of this iteration were to identify the scope of the genetic improvement tool, then—based on that scoping—identify web page design factors that have a potential relationship with the carbon footprint of a web page and develop fitness functions and code transformations based on those factors. The identification of fitness functions and actions was conducted by, first, performing a literature review of past work in this area. We then implemented actions and fitness functions and performed exploratory experiments to identify those that could be performed and measured reliably.

Scoping: A web page can consist of both *front-end* and *back-end* components, implementing user interfaces and underlying functionality, respectively. Front-end and back-end components can be built using many different programming or markup languages, and often make use of additional resources such as images, videos, or audio. Web page components can also be *localized* to a single computing unit or *geographically distributed*.

Our aspiration was to develop a tool that could be expanded over time to additional web page components, web programming languages, fitness functions, and code transformation actions. However, for the initial prototype, it was important to establish a clear scope that could be later enlarged.

In this study, we focus on **front-end, localized** web page components. Specifically, we modify **HTML, CSS, and JavaScript** components related to the user-facing interface of a web page. We also include **images** in this scope. In the current prototype, we focus on **the quantity of energy consumed**, and not the sources of energy (e.g., some geographic areas may use more renewable sources of energy than others).

Fitness Functions: To identify factors affecting a web page's carbon footprint, it is necessary to identify factors affecting its energy consumption. After examining past literature and exploring the feasibility of implementing measurement in a manner that (a) could be performed reliably[6], and (b) could be performed without specialized equipment[7], we implemented the following fitness functions:

- **Memory Usage:** Philippot et al. [17] identified a high correlation between memory and energy consumption for web pages. There is a correlation between the number of requests, the page size, and the consumed memory on the client. `Selenium`, a suite of tools for automating web browsers, is used to simulate the user environment of the web page to gather memory usage[8].
- **Quantity of Transferred Data:** Wholegrain Digital have developed a "Website Carbon Calculator" [5]. Their calculation is based on the quantity of data transferred to the end user, the energy intensity of the data, and the carbon intensity of the energy consumed. The latter two factors take into

[6] That yield relatively deterministic readings and are not heavily affected by the specific hardware and software configuration where the prototype tool is executed.

[7] Our desire was to develop a framework that could be used on any computer, without the need for dedicated equipment that measures energy consumption.

[8] https://www.selenium.dev/

account the locations where energy is consumed. As these are not considered in the current prototype, we focus solely on the quantity of data transferred. To measure this factor, we utilize the `PageSpeed Insights` API[9].

We also implemented, **but ultimately abandoned**, the following:

- **CPU Usage:** Many researchers use CPU usage to measure energy consumption, e.g., [4, 24]. However, after several experiments, we found that it was difficult to measure and isolate the CPU usage of the web page accurately. As a result, we decided to abandon this factor.

Reduction of carbon footprint must not negatively affect the user experience of the web page. Therefore, we also balance carbon footprint reduction against the following fitness functions, representing the user experience:

- **Page Load Time:** Load time is an important aspect of web page performance, as slow load times lead to many users leaving the page [19]. Load time and energy consumption are not strongly correlated [4]. Thus, we ensure performance is not negatively impacted while reducing the carbon footprint. We apply `Selenium` and `PageSpeed Insights` to capture load time.
- **Number of Changes:** The automated carbon footprint reduction tool should not overtly change the original code, as this may negatively affect the user experience, e.g., reducing usability or readability.

The prototype tool attempts to *minimize* the selected subset of fitness functions. This set of functions can be expended in future work as well. To address RQ1:

> **RQ1 (Factors):** The carbon footprint of a web page is affected by the quantity and location of energy consumed. We focused on quantity, and identified *memory usage* and *quantity of transferred data* as approximations that could be measured reliably and without specialized equipment.

Web Page Modifications: Through modification of code and resource elements, carbon footprint can potentially be reduced. These actions are intended to be applied to one compatible element at a time, e.g., a single image, HTML tag, or file. That way, the minimal set of actions that most strongly affect the carbon footprint can be identified. We have identified the following actions through analysis of literature and exploratory experimentation:

- **Change Image Format:** Energy consumption of image formats varies [22]. The WebP format has been found to yield smaller images than JPEG and PNG formats, leading to smaller data transmission quantity and improved page performance. Therefore, we convert images to WebP.
- **Compress Images:** Compressing images can reduce energy consumption [16]. Over-compressing images can reduce usability, but high image quality is not often required and users may not notice a difference after compression.
- **Swap HTML Tags:** `<strong>` elements use less energy and load faster than unstyled `<span>` tags [18]—the former requires 14% less loading time

[9] https://pagespeed.web.dev/

and energy, and `<span>` tags are particularly inefficient in mobile device browsers. Thus, we change the `<span>` tag to `<strong>` automatically.

- **Remove Unused CSS:** When a page is loaded, an HTML file is fetched and converted into a DOM object model. Afterwards, CSS stylesheets are fetched by the browser and converted to the CSSOM model [13]. A render tree is constructed by combining the DOM and CSSOM. The first content is produced by a browser only after this render tree has been constructed. Because of this, CSS files heavily affect the rendering time and quantity of data transferred [8]. When a CSS file contains unused CSS, it takes longer to build under the render tree. Thus, we remove unused CSS rules.
- **Remove CSS Opacity:** Translucent elements are rendered more slowly than opaque elements [18]. Consequently, removing opacity could potentially reduce energy consumption.
- **Move JavaScript Invocation:** Moving the JavaScript `<script>` to the end of the `body` can improve performance because the script invocation blocks parallel downloads, e.g., of image files [12].

We also implemented the following actions, but decided to discontinue their use following experimentation:

- **Remove HTML or CSS Whitespace:** Removing whitespace in the HTML or CSS source code can reduce the file size. This operation could be conducted before deploying the website to the server to avoid negatively impacting the readability of the code. However, due to this potential for reducing readability, we did not utilize this action in our initial prototype tool.
- **Use of `async` and `defer`:** The `async` attribute, used when invoking a JavaScript file in HTML, enables scripts to be loaded asynchronously. If a script uses this attribute, it will load independently and will not impede loading of other elements. Loading several scripts asynchronously can increase page loading speed and reduce total resource consumption. Similarly, when using the `defer` attribute, the browser will load the script after loading the page. This can also situationally improve resource consumption. However, we determined through initial experimentation that these changes could affect the behaviour of a web page in inadvertent ways.

Like with fitness functions, additional actions can be added in future work.

> **RQ1 (Factors):** The initial prototype attempts to reduce the carbon footprint by changing image formats, compressing images, swapping energy-consuming HTML tags for alternatives, removing unused CSS, removing opacity from elements, and moving JavaScript invocations.

3.2 Iteration II (Design of the Prototype Tool)

In the second iteration, we developed our prototype genetic improvement tool—making use of the fitness functions and actions developed in Iteration I. This tool performs multi-objective genetic improvement based on the NSGA-II algorithm (Non-Dominated Sorting Genetic Algorithm-II) [7]. The process followed by this tool is, as follows:

- An initial population of N solutions—alterations of the original web page—is generated by making random changes.
- This population is evolved over a series of generations. In each generation:
  - The best solutions in the population are identified (selection).
  - These solutions ("parents") are used to create a new "child" population, using the mutation and crossover operations.
  - The best N solutions from the parent and child populations form the population considered in the next generation, preventing the best solutions from being lost.
- After the final generation, the best solutions—those that cannot be dominated by any other solutions—are returned.

Parameters: The following parameters are user-adjustable:

- **Population Size:** The number of individuals in the population.
- **Search Budget:** The number of generations of evolution.
- **Fitness Functions:** The fitness functions to optimize. NSGA-II is generally considered efficient with up to three objectives [11]. Thus, we allow the selection of one to three fitness functions.

Solution Representation: Each individual represents an altered version of a web page. Solutions are represented using an array, where each item represents an action that can be applied to an element (e.g., compressing a particular image). The length of the array is fixed to the number of actions that could be applied to the particular page being optimized. A value of 0 means the original version of the element is kept, while a value of 1 means the element should be modified using the appropriate action.

Selection: NSGA-II uses the selection operator to select the most qualified candidates from the population as the basis for a new population. Each individual is evaluated according to how many of the others in the population dominate it and how many it dominates, measured using the set of fitness function. If no individual dominates another individual, that individual is considered to be in the first non-domination level, or Pareto frontier. To create the next population, individuals with higher non-domination ranks are selected, and if two individuals have the same non-domination rank, the selection operator favors the one with a greater crowding distance to preserve diversity. The crowding distance estimates the density of candidate solutions surrounding a particular individual in a Pareto frontier, with individuals from a less-dense region of the frontier preferred.

Tournament selection is used to select the parent population for creating the new child population. In a K-way tournament selection, K individuals are selected at random and compared from the previous population. The best two candidates are identified from this subset of the population.

Mutation and Crossover: At a certain probability, the mutation and crossover operations are applied to transform the "parent" solutions identified by the tournament selection into new, potentially improved, "child" solutions.

Mutation selects one action from a solution randomly and swaps the value in the array, i.e., replaces 0 with 1 or 1 with 0. Crossover creates two child solutions

Table 1: Project name and GitHub link for the final experimental subjects.

Project Name	Project Link
Poke-Dex	https://github.com/AM1CODES/Poke-Dex
Ecommerce-Website-main	https://github.com/MOUSTAFAAMIN25/Ecommerce-Website-main
htmlBurger-website	https://github.com/Marius-MPA/htmlBurger-website
complex-storm-html	https://github.com/kasumaputu06/complex-storm-html
module1-capstone-project	https://github.com/MahdiAghaali/module1-capstone-project
awesome-portfolio-websites	https://github.com/smaranjitghose/awesome-portfolio-websites
OpenWISP-Website	https://github.com/openwisp/OpenWISP-Website
project-website-template	https://github.com/yenchiah/project-website-template
ProjectSakura	https://github.com/ProjectSakura/ProjectSakura.github.io
Books-bootstrap-website	https://github.com/akashyap2013/Books-bootstrap-website

Table 2: Metadata on the web pages used for the final experiment.

Project Name	Num. Images	CSS Lines	HTML Lines	JS Lines
Poke-Dex	41	324	6713	111
Ecommerce-Website-main	33	275	415	319
htmlBurger-website	26	750	730	91
complex-storm-html	53	125	519	318
module1-capstone-project	10	206	192	83
awesome-portfolio-websites	2	1696	217	20
OpenWISP-Website	3	396	243	402
project-website-template	64	1841	1086	7
ProjectSakura	19	706	473	36
Books-bootstrap-website	6	422	355	51

by blending elements from the parent solutions. A simple form of crossover is utilized where two indexes are chosen at random, and the items at those indexes are swapped between the two parents. In the current prototype, the mutation probability is 0.4 and the crossover probability is 0.7. These values were identified through exploratory experimentation.

3.3 Iteration III (Refinement and Final Evaluation)

In the final iteration, we performed parameter tuning experiments for the tool, then used the identified parameter settings in a final evaluation. There are two main aims for conducting the evaluation. The first is to see if the prototype tool is able to reduce the carbon footprint of web pages. To this end, we compare to the *baseline of the initial fitness values for each web page with no changes made*. The second aim is to see if the prototype tool is more effective than picking transformation actions at random. The corresponding *baseline is imposing changes with a random generation tool*. That is, we generate a random solution and compare to the final solution produced by our tool.

Experiment Subjects: We evaluate our tool and the baselines using ten web page projects, selected from GitHub to represent varying degrees of complexity. Table 1 shows the project names and GitHub links, Table 2 presents metadata on each subject, and Table 3 shows the fitness values for the unmodified pages.

Experiment Configurations: We execute two configurations of our tool, each targeting three fitness functions: (1) *(Data Transferred Quantity) + (Number of Changes) + (Page Load Time)* (**Configuration DNT**), and (2), *(Data Transferred Quantity) + (Number of Changes) + (Memory Usage)* (**DNR**).

Table 3: Baseline fitness values of the web pages used for the final experiment.

Project Name	Data Transferred (KB)	Load Time (ms)	Memory Usage (KB)
Poke-Dex	5158908	3372	275528
Ecommerce-Website-main	3021120	1870	189700
htmlBurger-website	1907546	1937	163008
complex-storm-html	645713	2476	406188
module1-capstone-project	947203	1540	228976
awesome-portfolio-websites	843424	5802	238592
OpenWISP-Website	1238487	6077	172708
project-website-template	5141268	1743	273548
ProjectSakura	1422532	7052	223280
Books-bootstrap-website	1734210	2119	173412

The number of generation of evolution and population size found optimal by the parameter tuning were employed—a population of 20 solutions and a 20-generation search budget.

We perform 20 trials for each subject for each fitness function configuration. For every trial of our tool, we also perform one trial where the random baseline is executed. We refer to random trials paired with DNT-trials as "Random-DNT" and random trials paired with DNR-trials as "Random-DNR".

Data Collection and Analysis: We record the final value for each selected fitness function. Based on the collected data, we compared the performance of multi-objective optimization to the two baselines defined above first using the median fitness values for each fitness function and each experimental subject.

In addition, we compared the performance of the tool and the random baseline using statistical analysis for each fitness function. Data collected for each subject are drawn from an unknown distribution, and normality cannot be assumed. The Mann-Whitney Wilcoxon rank-sum test is used, with $\alpha = 0.05$, to determine whether the multi-objective tool and the random baseline yield fitness values drawn from different distributions. The following null hypothesis and alternative hypothesis are proposed for the analysis of the data collected:

- H0: Observations of results from the prototype tool are drawn from the same distribution as the random generation tool.
- H: Observations of results from the prototype tool are drawn from a different distribution than the random generation tool.

If the distributions are different, the Vargha-Delaney effect size test is used to measure the magnitude. We interpret effect sizes > 0.5 as the first technique performing better than the second. We follow the general interpretation, where an effect size of $0.56 \leq A < 0.64$ is classified as small, while $0.64 \leq A < 0.71$ is considered medium, and $A \geq 0.71$ is deemed large.

4 Results and Discussion

Median Fitness Values: Table 4 shows the median data transfer quantity for each baseline and the two configurations of the tool.

Performance (RQ2): In the median case, our tool reduces the quantity of data transferred by the original web page by 39.14% (DNT) and 40.68% (DNR) and by randomized changes by 25.42% (DNT) and 15.60% (DNR).

Table 4: Median quantity of data transferred (KB) for the original page, randomized baseline, and our tool. Lower values are better, and the lowest is bolded.

Project Name	Original	Random-DNT	Random-DNR	DNT	DNR
Poke-Dex	5158908	4942889	5069008	**4088047**	4181722
Ecommerce-Website-main	3021120	2136636	2167752	1477222	**1403839**
htmlBurger-website	1907546	1848460	1076644	928236	**881270**
complex-storm-html	645713	519497	444831	415664	**383898**
module1-capstone-project	947203	824963	824964	**803259**	818164
awesome-portfolio-websites	843424	767178	767155	**524879**	563573
OpenWISP-Website	1238487	766532	721795	736714	**716632**
project-website-template	5141268	4574004	4179609	4126205	**4110037**
ProjectSakura	1422532	1218049	1083724	**762353**	841990
Books-bootstrap-website	1734210	1090758	1026710	**651616**	1014127

Table 5: Median values for load time and memory usage for the original page, randomized baseline, and our tool. Lower values are better, the lowest is bolded.

	Load Time (ms)			Memory Usage (KB)		
Project Name	Original	Random-DNT	DNT	Original	Random-DNR	DNR
Poke-Dex	**3372**	3637	3526	275528	273394	**263079**
Ecommerce-Website-main	**1870**	1999	1883	189700	170972	**169447**
htmlBurger-website	1937	1870	**1836**	**163008**	165868	165013
complex-storm-html	2476	2474	**1873**	406188	444831	**402963**
module1-capstone-project	1540	1409	**1396**	228976	148444	**139848**
awesome-portfolio-websites	5802	6306	**5132**	238592	195604	**182057**
OpenWISP-Website	6077	985	**937**	172708	160578	**142629**
project-website-template	1743	1409	**1403**	273548	232752	**206689**
ProjectSakura	7052	6826	**6366**	223280	180134	**107218**
Books-bootstrap-website	2119	1859	**1855**	**173412**	182876	176933

Our tool's performance—generally for both fitness function configurations—is better than both baselines for all experiment subjects. The random baseline only attains comparable results for `OpenWISP-Website`. As discussed in Section 3.1, the quantity of transferred data is highly correlated with the carbon footprint of a web page [5]. Therefore, this is an indication that our tool can reduce a web page's carbon footprint significantly.

Table 5 shows the median memory usage for each baseline and the DNR configuration of the tool.

Performance (RQ2): In the median case, our tool reduces memory consumption of the page by 10.64% and of randomized changes by 3.96%.

Our tool yields lower memory consumption than both baselines for the majority of the subject projects, with the exception of the original versions of `htmlBurger-website` and `Books-bootstrap-website`. As memory usage is highly correlated with energy consumption [17], these results offers further evidence that our tool can reduce the carbon footprint of a web page.

The page load time was considered not because of a relationship with carbon footprint—past research found that it was not always correlated [4]—but because long load times can create a negative user experience [19]. We, therefore, minimize page load time to balance carbon footprint reductions with potentially negative changes to user experience. Table 5 also shows the median page load time for each baseline and the DNT configurations of the tool.

Table 6: Median number of changes made by randomized baseline and our tool.

Project Name	Random-DNT	Random-DNR	DNT	DNR
Poke-Dex	18.00	15.00	18.50	17.50
Ecommerce-Website-main	19.50	17.00	22.00	20.00
htmlBurger-website	13.50	12.50	16.50	16.00
complex-storm-html	9.50	9.50	12.50	13.00
module1-capstone-project	6.00	5.00	9.00	9.00
awesome-portfolio-websites	2.00	3.00	4.00	4.00
OpenWISP-Website	4.00	3.00	5.00	5.00
project-website-template	19.00	30.50	32.50	33.00
ProjectSakura	11.00	11.00	13.00	13.00
Books-bootstrap-website	4.00	8.50	12.00	13.00

Table 7: Effect sizes for data transfer quantity, load time, and memory usage in cases where a statistically significant difference exists between our tool and the random baseline. Large effect sizes are in bold.

Project Name	Transfer(DNT)	Transfer (DNR)	Load Time	Memory Usage
Poke-Dex	**0.99**	**0.88**	**0.96**	**0.76**
Ecommerce-Website-main	**0.93**	**0.88**	**0.80**	**0.76**
htmlBurger-website	**0.82**	**0.84**	**0.77**	-
complex-storm-html	**0.88**	**0.84**	**0.81**	**0.75**
module1-capstone-project	**0.96**	**0.93**	**0.91**	**0.87**
awesome-portfolio-websites	**0.89**	**0.94**	**0.80**	**0.87**
OpenWISP-Website	-	**0.78**	0.70	**0.88**
project-website-template	**0.93**	**0.99**	**0.75**	**0.85**
ProjectSakura	**0.91**	**0.82**	**0.83**	**0.77**
Books-bootstrap-website	**0.95**	**1.00**	**0.77**	**0.91**

Performance (RQ2): In the median case, our tool reduces the load time of the original page by 14.05% and of randomized changes by 6.36%.

We can see that our tool also yields lower load times for the majority of the experiment subjects, with the exception of `Poke-Dex` and `Ecommerce-Website-main`. In other words, in many cases, reductions in data transfer quantity can also yield an improved user experience through faster loading times.

However, we can also see from the deviating cases that improvements to these fitness functions do not always correlate. Changes intended to reduce the data transferred can reduce memory consumption and load time as well, but they can also have a slight negative impact in some cases.

Table 6 shows the median number of changes made by the randomized baseline and our tool. Our tool tries to balance improvements in other fitness functions against the number of changes made to preserve the usability of the page. More changes are made by our tool than by random generation in the average case. However, for many pages, the set of changes is still relatively small.

Statistical Analysis: Table 7 shows that there is a statistical difference in the data transfer quantity between our tool and random solution generation for almost all subjects, with the exception of `OpenWISP-Website` under the DNT configuration. In all cases where such a difference exists, the DNT configuration outperforms that random baseline with a large effect size. In all ten subjects, the DNR configuration outperforms the random generation with a large effect size with regard to quantity of data transferred.

Table 8: Effect size results for data transfer quantity when comparing the two fitness function configurations. Large effect sizes are in bold.

Project Name	Effect Size
Poke-Dex	**0.07**
Ecommerce-Website-main	-
htmlBurger-website	**0.89**
complex-storm-html	**0.92**
module1-capstone-project	**0.03**
awesome-portfolio-websites	**0.00**
OpenWISP-Website	**1.00**
project-website-template	**0.73**
ProjectSakura	**0.27**
Books-bootstrap-website	**0.00**

With regard to load time and memory usage, Table 7 shows that statistical differences exist for almost all subject projects, with the exception of the memory usage of `htmlBurger-Website`. Our tool outperforms the random baseline in each case where a statistical difference was observed. In terms of page load time, the tool outperforms the random baseline with a large effect size for nine projects and one with a medium effect size. In terms of memory usage, the tool outperforms the random baseline with a large effect size for nine projects.

Performance (RQ2): Our tool outperforms the random baseline in 95.00% of comparisons with statistical significance—with a large effect size in 92.50% of comparisons.

We can also compare the two configurations of our tool in terms of the quantity of data transferred—a fitness function shared in both configurations. Table 8 shows that there are statistically significant differences in the results in nine of the ten projects. However, there is not a clear pattern in *which* configuration is better. In four of ten projects, the DNR fitness function combination outperforms the DNT fitness function combination with a large effect size. However, in the other five projects where a distribution difference was detected, the DNT fitness function combination outperforms the DNR configuration.

It is not clear exactly why one configuration outperforms another in these cases with regard to the quantity of data transferred. It may be that pursuit of one of the other fitness functions—memory usage or load time—may offer feedback on how to further reduce the quantity of transferred data. Future research should explore both additional fitness functions as well as different combinations of fitness functions to identify the tool configurations most widely effective.

Modifications Used: We also examine which modifications are applied most often. Table 9 shows the percentage of final solutions where a particular type of action has been taken for at least one compatible element when targeting the DNT configuration. Table 10 shows the same for the DNR configuration.

As might be anticipated, changes to the images—both compression and format changes—are applied particularly often and clearly have an impact on the quantity of transferred data, memory usage, and page loading time. However, the other actions are applied as well. In particular, unused CSS is frequently

Table 9: The percentage of final solutions that apply an action of a particular type to at least one compatible element for the **DNT** configuration.

Project Name	Move Script(%)	Remove Opacity(%)	Change HTML(%)	Remove Unused CSS(%)	Compress Image(%)	Convert Image(%)
Poke-Dex	25.00	75.00	65.00	80.00	100.00	100.00
Ecommerce-Website-main	30.00	55.00	55.00	60.00	100.00	100.00
htmlBurger-website	35.00	35.00	55.00	65.00	95.00	100.00
complex-storm-html	45.00	40.00	50.00	50.00	100.00	100.00
module1-capstone-project	35.00	50.00	75.00	80.00	100.00	100.00
awesome-portfolio-websites	25.00	25.00	40.00	90.00	100.00	100.00
OpenWISP-Website	45.00	25.00	65.00	50.00	100.00	100.00
project-website-template	15.00	15.00	60.00	75.00	100.00	100.00
ProjectSakura	10.00	40.00	70.00	70.00	100.00	100.00
Books-bootstrap-website	25.00	55.00	55.00	90.00	100.00	100.00
Overall	28.50	52.00	59.00	71.50	99.50	100.00

Table 10: The percentage of final solutions that apply an action of a particular type to at least one compatible element for the **DNR** configuration.

Project Name	Move Script(%)	Remove Opacity(%)	Change HTML(%)	Remove Unused CSS(%)	Compress Image(%)	Convert Image(%)
Poke-Dex	15.00	40.00	45.00	80.00	100.00	100.00
Ecommerce-Website-main	35.00	50.00	50.00	55.00	100.00	100.00
htmlBurger-website	30.00	50.00	65.00	55.00	100.00	100.00
complex-storm-html	15.00	60.00	65.00	70.00	100.00	100.00
module1-capstone-project	55.00	40.00	50.00	50.00	95.00	100.00
awesome-portfolio-websites	40.00	30.00	40.00	65.00	100.00	100.00
OpenWISP-Website	30.00	15.00	70.00	65.00	100.00	100.00
project-website-template	25.00	30.00	50.00	80.00	100.00	100.00
ProjectSakura	15.00	15.00	55.00	65.00	100.00	100.00
Books-bootstrap-website	10.00	50.00	65.00	80.00	100.00	100.00
Overall	26.50	43.00	57.00	62.50	99.50	100.00

removed, perhaps because many websites make use of existing templates and their creators do not optimize the templates. Addressing RQ3:

Modifications (RQ3): Compressing, converting images and removing unused CSS are the most common modifications applied by our tool.

5 Threats to Validity

Conclusion Validity: When using statistical analyses, we have attempted to ensure the base assumptions behind these analyses are met. We have favored non-parametric methods, as distribution characteristics are not generally known a priori, and normality cannot be assumed.

To control experiment cost, we have only performs twenty trials for each tool configuration and case example. It is possible that larger sample sizes may yield different results. However, given the consistency of our experiment results, we believe that this is a sufficient number of repetitions to draw stable conclusions.

External Validity: Our results are specific to HTML and CSS, and our techniques and findings cannot be expected to map to additional languages or file formats. Our study has also focused on ten subject web pages—a relatively small

sample. Nevertheless, we attempted to identify open-source web pages representing a range of sizes and use cases. We believe that our subjects are generally representative of small-to-medium-sized web pages.

6 Conclusions

We have explored the use of genetic improvement to automatically reduce the carbon footprint of web pages. In the average case, our tool was able to reduce the quantity of data transferred for all projects, and the memory consumption and loading time for eight of the projects. Our tool outperformed the random baseline with large effect size in 92.50% of comparisons. Compressing and converting images and removing unused CSS were the most common actions performed by the tool, with major effects on solution fitness.

In future research, we will expand the range of actions and fitness functions, as well as the scope of experiment subjects considered. We will also perform a user study to qualitatively assess the acceptability of the applied transformations. We are also particularly interested in examining the impact of the location where components are hosted on the carbon footprint of web applications. We will also explore ways to encourage users to regularly use and trust such tools as part of their workflow.

References

1. A. S. Andrae. New perspectives on internet electricity use in 2030. *Engineering and Applied Science Letters*, 3(2):19–31, 2020.
2. T. Bawden. Global warming: Data centres to consume three times as much energy in next decade experts warn. Available at: https://www.independent.co.uk/climate-change/news/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html, 2016. Accessed: 12.11.2022.
3. B. R. Bruce, J. Petke, and M. Harman. Reducing energy consumption using genetic improvement. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1327–1334, 2015.
4. Y. Cao, J. Nejati, P. Maguluri, A. Balasubramanian, and A. Gandhi. Analyzing the power consumption of the mobile page load. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, pages 369–370, 2016.
5. T. F. Chris Adams, Rym Baouendi. Website carbon calculator. 2022.
6. V. De La Luz, M. Kandemir, and I. Kolcu. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *Proceedings 2002 Design Automation Conference (IEEE Cat. No. 02CH37324)*, pages 213–218. IEEE, 2002.
7. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.
8. C. Developers. Remove unused css. Available at: https://developer.chrome.com/docs/lighthouse/performance/unused-css-rules/, 2020. Accessed: 09.11.2022.

9. J. Dorn, J. Lacomis, W. Weimer, and S. Forrest. Automatically exploring tradeoffs between software output fidelity and energy costs. *IEEE Transactions on Software Engineering*, 45(3):219–236, 2017.
10. K. Haan. How many websites are there? Available at: https://www.forbes.com/advisor/business/software/website-statistics/, 2023. Accessed: 04.10.2022.
11. V. Khare, X. Yao, and K. Deb. Performance scaling of multi-objective evolutionary algorithms. In *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings 2*, pages 376–390. Springer, 2003.
12. Kirupa. Running your code at the right time. Available at: https://www.kirupa.com/html5/running_your_code_at_the_right_time.html, 2020. Accessed: 01.12.2022.
13. L. Lazaris. An introduction and guide to the css object model (cssom). Available at: https://css-tricks.com/an-introduction-and-guide-to-the-css-object-model-cssom/, 2018. Accessed: 08.11.2022.
14. I. Manotas, L. Pollock, and J. Clause. Seeds: A software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering*, pages 503–514, 2014.
15. V. Mrazek, Z. Vasicek, and L. Sekanina. Evolutionary approximation of software for embedded systems: Median function. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 795–801, 2015.
16. O. Philippot. Which image format choose to reduce energy consumption and environmental impact? Available at: https://greenspector.com/en/which-image-format-to-choose-to-reduce-its-energy-consumption-and-its-environmental-impact/, 2022. Accessed: 15.01.2023.
17. O. Philippot, A. Anglade, and T. Leboucq. Characterization of the energy consumption of websites: Impact of website implementation on resource consumption. In *ICT for Sustainability 2014 (ICT4S-14)*, pages 171–178. Atlantis Press, 2014.
18. A. Sampson, C. Caşcaval, L. Ceze, P. Montesinos, and D. S. Gracia. Automatic discovery of performance and energy pitfalls in html and css. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 82–83. IEEE, 2012.
19. W. Stadnik and Z. Nowak. The impact of web pages' load time on the conversion rate of an e-commerce platform. In *International Conference on Information Systems Architecture and Technology*, pages 336–345. Springer, 2017.
20. S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, pages 409–415. IEEE, 2002.
21. J. Taina. How green is your software? In *International Conference of Software Business*, pages 151–162. Springer, 2010.
22. N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery? analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50, 2012.
23. D. R. White, J. Clark, J. Jacob, and S. M. Poulding. Searching for resource-efficient programs: Low-power pseudorandom number generators. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1775–1782, 2008.
24. Y. Zhu and V. J. Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24. IEEE, 2013.