AUTO-DESIGN OF WEB PAGES USING AN INTERACTIVE GENETIC ALGORITHM

I. Introduction

A genetic algorithm (GA) is a computational method used in computer science to help develop the best solution to a problem [1]–[3]. Its foundation is rooted in the theory of biological evolution as proposed by Charles Darwin in his famous 1859 book *Origin of Species* [1], [4]. A GA consists of four basic processes. The first is initialization, which creates the initial candidate solutions; the second is evaluation, which rates how well each member of the initial group solves the problem at hand; the third is selection, which chooses adequate solutions from the candidate pool and discards the others; and the fourth is recombination, or reproduction, which takes the solutions that survived the selection process and combines them to form new candidate solutions that will hopefully produce a better solution to the problem than the previous generation [1]. The final three processes are repeated until a specific end condition is met, which is typically when the algorithm produces a solution that solves the problem. The GA was developed by John Holland in 1962 while the concept of genetic programming, which is the application of GAs to problem solution, was developed by John Koza in 1992 [1], [4], [5]. Koza wrote that genetic programming is "problem independent," which means that these four basic processes can be used to solve any problem [2].

An interactive genetic algorithm (IGA) differs from a traditional genetic algorithm during the evaluation process. The user decides the fitness of the candidate solutions instead of the algorithm. Interactive genetic algorithms are used when a solution cannot be found mathematically, such as when what is considered a good solution is entirely subjective [6]. Users must be able to easily evaluate such candidates, which implies that they must be visualized [6]. IGAs can present possibilities to the user that he or she may not have considered [6].

II. Process Overview

The interactive genetic algorithm is a PHP web application (web app) that consists of three main steps: initialization, evaluation, and recombination. The initialization step creates candidates based upon default or user-selected initial settings. The evaluation step displays screenshots of the candidates to the user as thumbnail images. The recombination step, which is the most important, generates new candidates to replace those not selected by the user through the use of simulated genetic operators at fixed rates. If no operator is used, the IGA generates a candidate from "scratch" via initialization instead.

The general outline of the web app is as follows:

1. Give the user a chance to change initial settings.
2. Randomly generate a candidate population.
3. Allow the user to evaluate candidates.
4. Perform recombination on the current population.
5. Repeat steps 3 and 4 until the user settles on a single candidate, decides to start over, or quits.

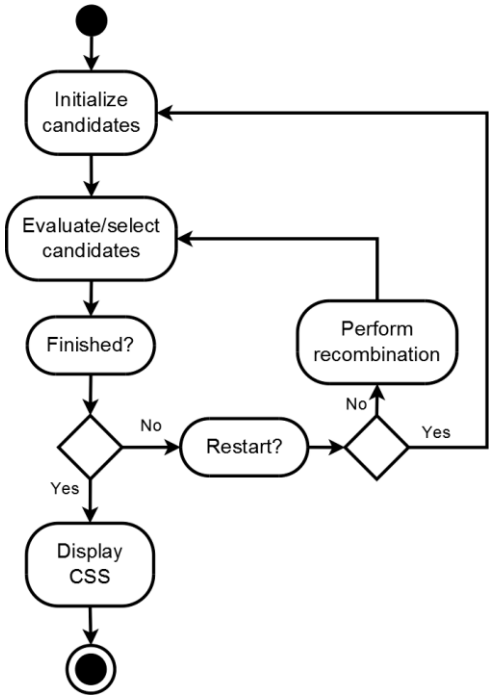


Fig. 1. A visual overview of the IGA process.

The web app evolves style sheets used to define the visual appearance of a web page that is mostly static. Cascading Style Sheets (CSS) is a scripting language consisting of selectors that indicate which portions of a web page to style and properties that modify the attributes of selectors. The web app dynamically creates a style sheet for each candidate.

While there is no guarantee that the IGA will produce the optimal solution, the size of the search space will allow it to present a large variety of different designs to the user. There is a total of 20 CSS selectors (ID selectors as well as HTML element selectors such as "body" and "p") (see Table I) and a total of 27 CSS properties (see Table II) in use that can be divided into two categories: visual and text. Not all properties are applicable to every selector, however. For example, only properties that alter visual characteristics are used for images. Unrestricted, the search space is a staggering $1.6x10^{343}$ style sheets. However, to guarantee specific layouts, some properties are restricted and their value either depends upon what was selected in the initialization settings or upon the value generated for a

previous property. The size of the search space, when taking into account restrictions, ranges from $5.3x10^{81}$ to $7.8x10^{313}$ style sheets. Even though it is always very large, it is still possible for the web app to generate satisfactory designs because it takes into account the user's preferences.

TABLE I
PAGE ELEMENT SELECTORS

Selector	Description
<body>	Web page body.
#container	Web page container.
#header	Web page header section.
#nav	Web page navigation section.
#sidebar	Web page sidebar section.
#content	Web page content section.
#footer	Web page footer section.
<h1>	1st level heading.
<h2>	2nd level heading.
<h3>	3rd level heading.
<p>	Paragraph.
<hr>	Horizontal rule (section divider).
	Unordered (bulleted) list.
	List item.
<img>	Image.
<a>	Anchor (link).
:link	Unclicked link pseudoselector.
:visited	Visited link pseudoselector.
:hover	Mouse pointer hover link pseudoselector.
:active	Mouse click link pseudoselector.

TABLE II
CSS PROPERTIES

Visual Layout	Text Layout
background-color	color
border color	font-family
border-style	font-size
border-width	font-style
clear	font-variant
display	font-weight
float	letter-spacing
height	text-align
line-height	text-transform
margin-bottom	
margin-left	
margin-right	
margin-top	
padding-bottom	
padding-left	
padding-right	
padding-top	
width	

III. INITIALIZATION

Initialization begins with a configuration page (see Fig. 2). There are five categories of options available for the user to customize: the overall page layout; the main color scheme; visual layout properties such as width and height; border; and text properties. A special value, "random", allows the web app to randomly select from a list of acceptable values while "inherit" instructs the browser to use the value of an ancestral element for that property.

There are five page layouts to choose from or the user can allow the script to randomly choose. Layouts #4 and #5 have several variations due to the possible ordering of the columns. The layouts are examples of styles that actual websites use (see Fig. 3). There is a total of five page sections that can be styled, but not every layout makes

use of them all. Only four are typically used: the header, the footer, the navigation bar (nav bar), and the main content area. The fifth section, the sidebar, is only used by a few types of websites (such as online stores) as a secondary navigation bar or to provide additional information to the user.

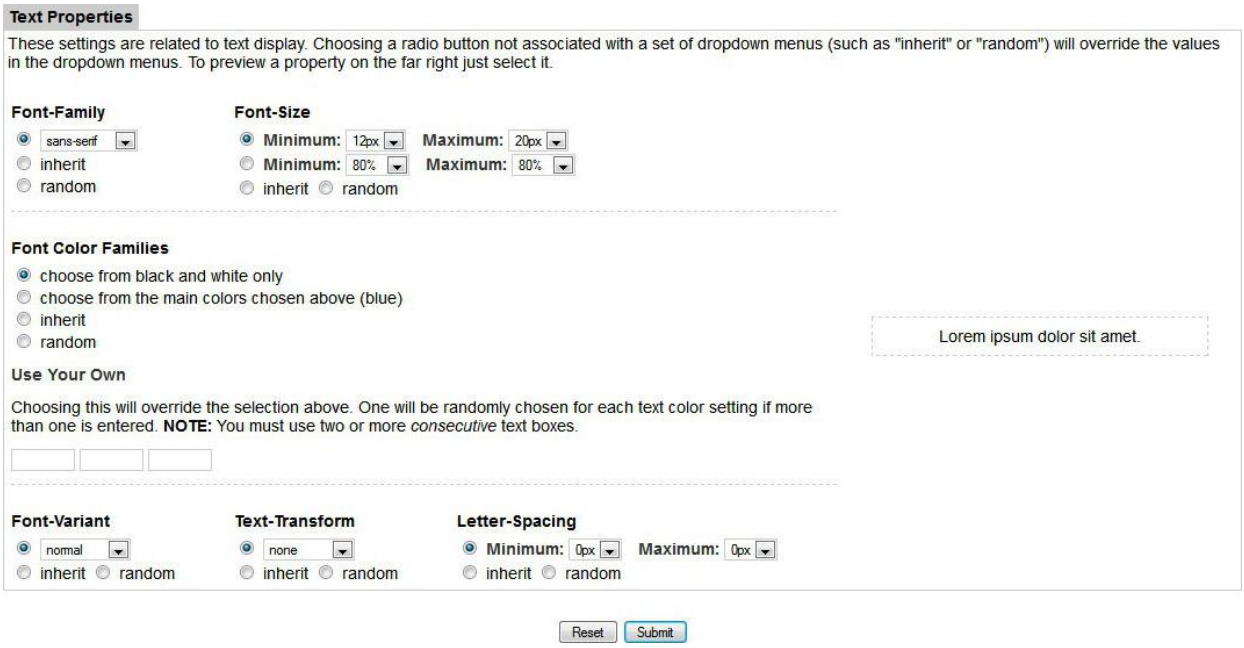
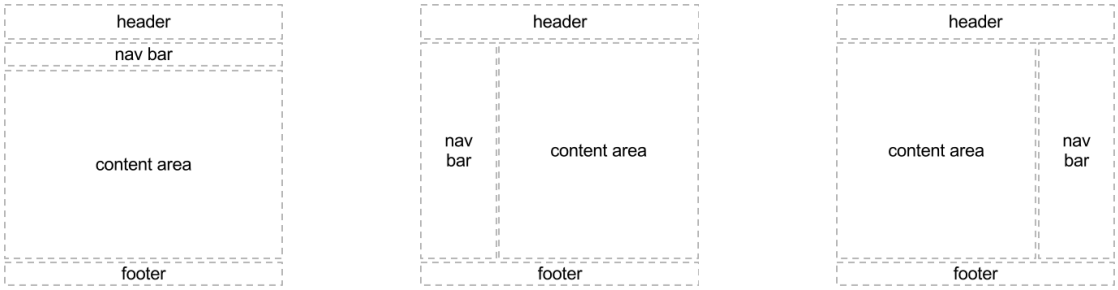


Fig. 2. A portion of the configuration page.



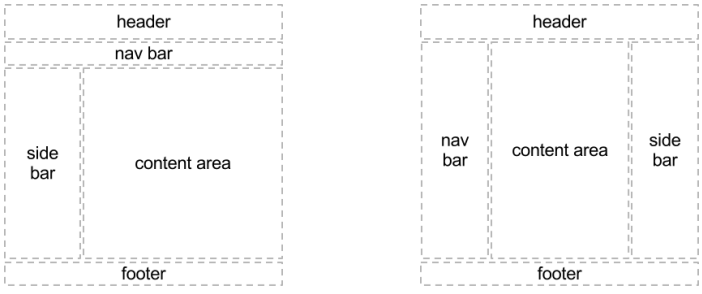


Fig. 3. The five layout styles.

Each chromosome, or design candidate, is represented as a single string of 1s and 0s that are CSS property values and value types translated into binary from other values (integer, hexadecimal, and string) and concatenated together in a specific order. Each selector's associated properties are also concatenated together to form the chromosome as a whole. Every chromosome has the same number of selectors, but not every selector has the same number of properties. The property's value becomes the first portion of the property's section of the bit string. The second portion, the two bit value type, is what differentiates two otherwise equivalent integer values of two different property types. It distinguishes 10 pixels from 10 percent, for example.

(a) {[properties of <body>][properties of #container] ... [properties of :active]}

(b) {[bits 0-134][bits 135-273] ... [bits 3543-3706]}

(c) [(background-color)(margin-top) ... (text-align)]

(d) [(bits 0-25)(bits 26-31) ... (bits 130-134)]

Fig. 4. Bit string organization for a chromosome. (a) Organization of selected CSS selectors. (b) Bit ranges of selected CSS selectors. (c) Organization of selected properties of <body>. (d) Bit ranges of selected properties of <body>.

IV. Evaluation

The user evaluation step is simple. Enlargeable thumbnail images of each candidate are displayed to the user in a table. Once he or she is finished selecting preferred candidates using checkboxes, one of the following occurs: (1) selected candidates are used to create new candidates to replace those not selected, (2) the user makes a final selection, or (3) the user starts over. The total number of candidates presented to the user can vary. If the height of his or her resolution is less than 768 pixels, the candidate pool will contain 15 candidates; otherwise, it will contain 20. When the evaluation step is repeated, only the candidates that were selected during the most recent round of evaluation and the new candidates created during recombination will be displayed to the user.

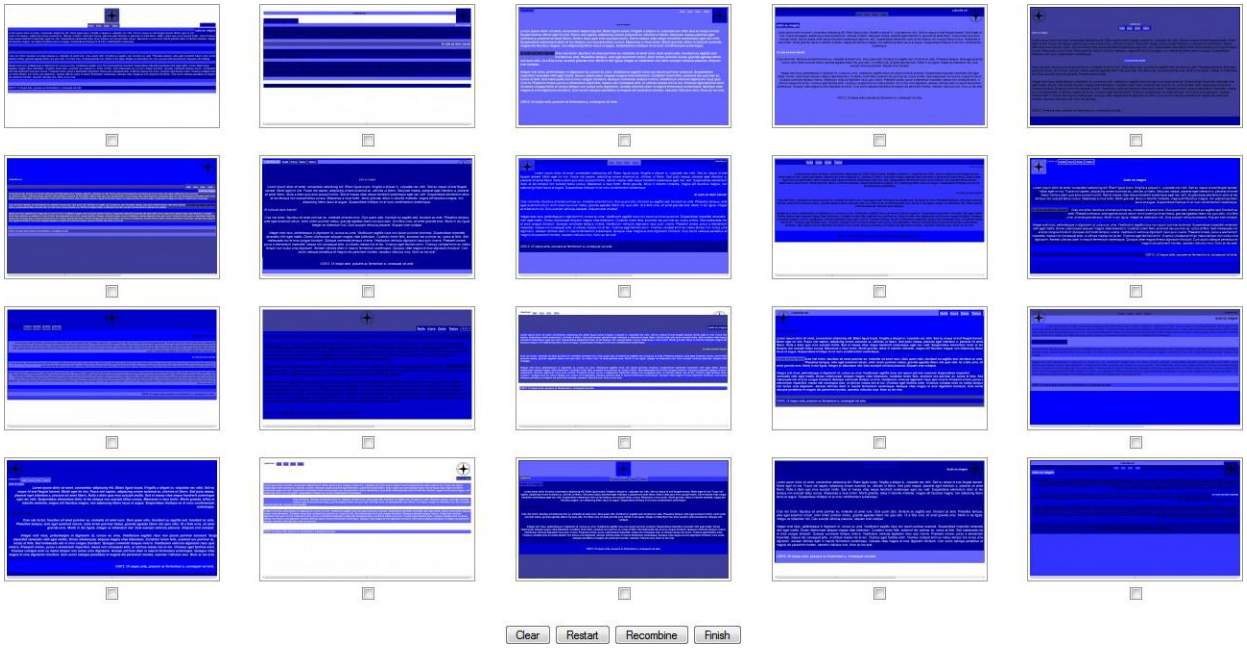


Fig. 5. Example screenshot of the evaluation page.

V. RECOMBINATION

During recombination, new candidates are created to replace those the user did not select. If the user clicked "finish" on the evaluation page, the candidate that he or she selected as their final choice is then displayed. If the user clicked "restart", the initialization settings page is displayed. Otherwise, the recombination process begins.

It can be described as follows:

1. Record the fitness of each candidate.
2. Check the number of selected candidates.
   a. If one candidate is selected, repeatedly perform mutation on it until the current population size equals the maximum size *n* of the candidate pool.
   b. If between two and *n*-1 candidates are selected, create a probability distribution that assigns a selection probability to each candidate that is proportionate to its fitness. Perform multipoint crossover and/or mutation on candidates selected via roulette-wheel selection. Possibly perform mutation on children created via crossover. As a last resort, generate a brand new candidate. Do this until the current population size equals *n*.
   c. If all candidates are selected, propagate the entire generation.
   d. If no candidates are selected, recreate the entire generation.

After each round of user evaluation, the number of times a candidate has survived evaluation is increased by one and the value for the candidates that did not survive is reset to zero. This "survival value" is also equal to the fitness of each candidate. The fitness determines the probability of a candidate being selected for crossover.

The probability distribution is created by dividing each candidate's fitness by the sum of all the fitness values of the candidates selected by the user. Ideally, the roulette values of all selected candidates added together

would equal "1", but it is sometimes necessary to adjust the values to make this true by allocating the remainder of the distribution value (1-*sum*) to the candidate whose fitness value only appears once in the population. If this is not possible, the maximum value of the distribution is defined as the unadjusted value and a small margin of error is introduced. This distribution sets the values of the roulette wheel used to select parents for crossover.

New candidates are created through the use of the crossover and/or mutation operators at a rate of 0.6 and 0.05 respectively, or through initialization. If only one candidate was selected by the user during evaluation, it is possibly mutated to create a new candidate by comparing a randomly generated floating point number to the mutation probability p_M. If the random number is greater than p_M, the new candidate is initialized instead.

If a candidate was selected for mutation, a bit in its bit string is randomly selected and then subsequently flipped (a "1" becomes a "0" and a "0" becomes a "1"). If the bit that was flipped was part of a property type portion of the string instead of a value portion, it is sometimes necessary for the value to change to match the new type and its associated value restrictions (see Fig. 6). Certain properties of certain selectors are not allowed to be mutated in order to preserve the page layout.

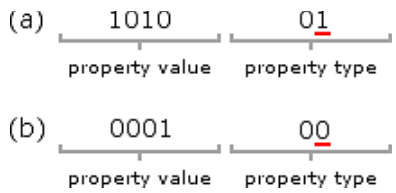


Fig. 6. Example of property type change caused by mutation. (a) Before mutation (the value is 10 pixels). (b) After mutation and correction (the value is now "inherit", a string). The flipped bit is underlined.

If the number of candidates selected is between two and *n*-1, two candidates are chosen using roulette-wheel selection and possibly mated. A floating point number is randomly generated and then compared to each value on the roulette wheel until a candidate is selected. If the random number is less than or equal to the roulette value of the candidate currently being tested, the previous roulette value is then compared to the random value. If it is greater than this number, the current candidate is selected. For example, if 0.13 is generated, candidate #1 is selected (see Fig. 7). This process is then repeated for the selection of the second parent. If the first parent is selected a second time, the process is repeated until both parents are unique.

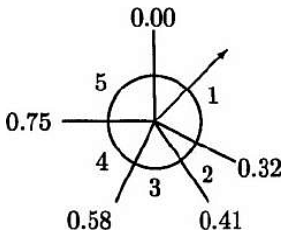


Fig. 7. An example of roulette-wheel selection [7].

The crossover process clones two parent candidates, randomly chooses two points in each bit string, and swaps the bits between these points so that two new child candidates are created. If the length of the bit string is *len*, the selection of the first point can be made from any position in the range of 0 to *len-1* (inclusive). The selection of the second point, however, is much more restricted. It must be carefully controlled so that all bit string swaps occur within the boundaries of a single element selector. Fig. 8(a) shows two crossover points that fall within the boundaries of both the <body> (bits 0-134) and #container (bits 135-273) selectors while Fig. 8(b) shows the points after they have been adjusted to fall within the boundary of only the <body> selector.

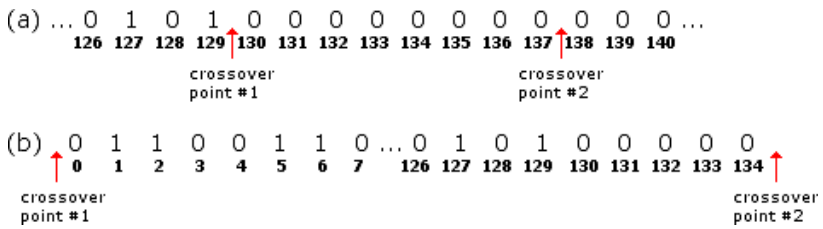


Fig. 8. An example of crossover point selection. (a) Before adjustment. (b) After adjustment.

If the two crossover points are identical, the second point is selected again until it is unique. Next, the points are sorted in ascending numerical order. Finally, the values corresponding to the selector and properties in the previously cloned CSS arrays are also swapped.

The two new children are initialized with the altered bit strings, CSS files are created from the altered CSS arrays, and screenshots are taken. Each child is then possibly mutated in an attempt to introduce greater variety into the candidate population.

If the randomly generated floating point number is greater than the crossover probability p_C, a candidate is randomly selected for possible mutation instead. If mutation fails, then a brand new candidate is initialized. Testing for crossover continues until either all available positions are assigned to new candidates or the number of available

positions equals one. If there is only one position remaining, a new candidate is created using either mutation or initialization.

If every candidate was acceptable to the user, the current generation is propagated to the next with no alteration. If no candidates were acceptable, the entire population is discarded and another is created via initialization.

V. Finishing

Once the user has settled on a final design by selecting "finish" on the evaluation page, a simple "finishing" script is executed. The user is unable to undo this action once completed. If he or she wishes to explore more design possibilities, he/she has no choice but to start over from the beginning. The design is then displayed to the user beside the CSS that created it.

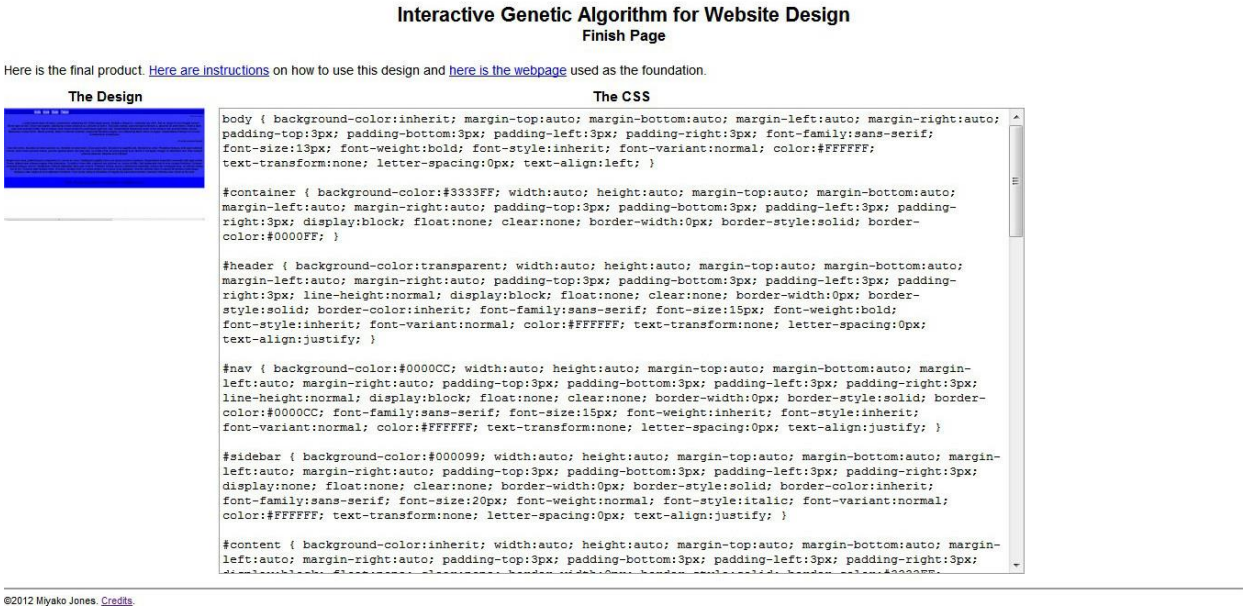


Fig. 9. An example of the finish screen.

VI. Conclusion

Though there is certainly room for improvement, the IGA successfully generates a variety of candidates for the user to choose from. The user's preferences are taken into account from start to finish and he or she is directly involved in the evolution of the candidate population. Although the "perfect" web page design may never be generated, the user can explore many others that might help him or her develop their ideal design.

REFERENCES

[1] M. T. Jones. (2003). "Introduction to Genetic Algorithms," in *AI Application Programming (1st ed.)* [Online]. Available: http://site.ebrary.com/lib/umich/Doc?id=10061221

[2] J. R. Koza. (2007, Jul. 8). Genetic Programming, Inc. homepage. [Online]. Available: http://www.genetic-programming.com

[3] M. Srinivas and L. M. Patnaik. (1994, Jun.). Genetic algorithms: A survey. *IEEE Computer* [Online]. *27(6),* pp. 17-26. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=294849&isnumber=7284

[4] C. R. Reeves and J. E. Rowe. (2002). "Introduction," in *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory (1st ed.)* [Online]. Available: http://site.ebrary.com/lib/umich/Doc?id=10067477

[5] A. Oliver, N. Monmarch, and G. Venturini. "Interactive design of web sites with a genetic algorithm," presented at IADIS International Conference WWW/Internet. [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.1950

[6] N. Monmarché, et al. "Imagine: A tool for generating HTML style sheets with an interactive genetic algorithm based on genes frequencies," presented at SMC'99. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.6918

[7] C. R. Reeves and J. E. Rowe. (2002). "Basic Principles," in *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory (1st ed.)* [Online]. Available: http://site.ebrary.com/lib/umich/Doc?id=10067477