ELSEVIER

Computer Networks 36 (2001) 311–321

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Optimization of web newspaper layout in real time

J. González, I. Rojas *, H. Pomares, M. Salmerón, A. Prieto, J.J. Merelo

Department of Computer Architecture and Computer Technology, University of Granada, Campus de Fuentenueva, E. 18071 Granada, Spain

Received 23 March 2000; received in revised form 20 October 2000; accepted 17 November 2000

Responsible Editor: M. Hamdi

Abstract

The web newspaper pagination problem consists in optimizing the layout of a set of articles extracted from several web newspapers and sending it to the user as the result of a previous query. This layout should be as similar as possible to that of real newspapers and should be adapted to the user's web browser configuration in real time. Previous approaches to the problem took rather a long time or suffered the constraint of all the articles having the same width. This paper presents an improved approach based on simulated annealing (SA) that solves the problem online, adapts itself to the client's computer configuration and also supports articles with different widths, so that long articles can now be laid out in more than one column, producing a squarer shape. With this new feature, the layouts obtained look more like a real newspaper, where important or long articles occupy more than one column.

Keywords: Simulated annealing; Web newspaper; Real-time optimization

1. Introduction

Since the amount of information available on the Internet is growing day by day, when a user sends a query to a news site, a lot of information is returned and the user frequently has great difficulty in finding the piece of information he really wants. In the case of web newspapers it would be desirable for the final look of the resulting information to be as similar as possible to that of real newspapers. The articles should be organized in columns with no overlap so that the user can read all of them, and should occupy the smallest possible area, minimizing gaps between articles, and avoiding scroll bars if possible to facilitate reading.

As the query is sent via a web browser, the results should also be presented as a web page and as soon as possible. This requirement transforms the layout optimization task into a real-time problem, thus requiring a fast optimization method to solve it. The algorithm proposed in this paper solves the problem rapidly compared with the time spent loading the web page to be optimized, while taking into account the font sizes and faces the user is using and the size of the web browser window where the result is to be displayed, thus obtaining a

* Corresponding author. Tel.: +34-958-246128; fax: +34-958-248993.

E-mail addresses: jesus@ugr.es (J. González), irojas@atc.ugr.es (I. Rojas).

1389-1286/01/$ - see front matter
PII: S1389-1286(01)00158-X

result that is adapted to the client's computer configuration.

There are two ways to tackle this problem. The first one is to solve it at the server side, when the user's query is received. This approach obtains a generic result that might not be well suited to the user's web browser configuration. Another disadvantage is that if the news server has lots of queries to solve at the same time, it could become overloaded.

On the other hand, if the algorithm that performs the layout optimization is sent in the same web page to be optimized as a script that will be interpreted in the web browser when page loading finishes [4,5], the server only has to look for the articles that match the user's query and send them, this being much faster than the above approach. Optimization takes place in the client's computer web browser, just after the web page containing the articles and the optimization algorithm has been loaded, taking into account the web browser window dimensions where the information is to be displayed and the font faces and sizes adopted by the user, thus obtaining a result that is adaptable to every possible client web browser configuration without overloading the server.

This paper describes a SA-based algorithm for the web pagination problem that optimizes the layout of all the articles matching the client's query to a web newspaper in real time and performs all the optimization tasks within the client's computer. The algorithm is able to manage articles with several widths and generates a web page that is very similar to a real newspaper, taking into account the size of the browser window and the face and size of the fonts in the client's machine, thus producing a layout that adapts itself to the user's computer characteristics.

This paper is organized as follows: In Section 2 some literature concerning the problem of web newspaper layout optimization is described. The proposed approach to tackle this issue is discussed in Section 3. In Sections 4 and 5 the main algorithm parameters are analyzed and some results are presented. Finally several possibilities for future work are mentioned in Section 6 and final conclusions are drawn in Section 7.

2. State of the art

There is not much literature about the pagination problem, and almost all the studies reported to date are offline approaches to the pagination problem, where the time taken to solve the problem is not very important. One example of such an approach is the *YPSS++* system [6] used to paginate Yellow Pages.

A group of researchers in the Finnish Research Institute VTT has applied simulated annealing (SA) to optimize fax newspapers and Yellow Pages in several countries [11]. In their paper, three different approaches to the problem are presented, a heuristic method and two others that use SA. The methods using SA obtain better results, but overlapping between articles is allowed and even with the best of the SA methods a slight overlap of articles in the final result is sometimes observed.

The field of online approaches to the pagination problem includes, for example, the *Krakatoa Project* [8], which implements a personalized newspaper as a *Java* applet embedded in a web page and is able to customize the final layout depending on the user profile, although it does not optimize the total surface occupied.

Another approach is described in [4], where the articles extracted from several newspapers as the result of a user query are displayed on a web page; an attempt is made to provide a newspaper-like layout. This approach is based on a genetic algorithm (GA) [3,7,13] where the top-left corner of each article is encoded as a gene in every chromosome in the population. This representation is not very good because it requires a lot of calculation to check whether two or more articles are overlapping, making the algorithm very slow.

Another approach based on SA [5] uses a different representation that obtains better results (10 times faster than in [4]), but with the constraint that all articles have the same width. Thus, the surface of the web browser window is divided into several columns of a fixed width. Possible solutions are represented as permutations of articles to be displayed using a greedy algorithm that allocates the articles in the least occupied column according to the order they have in the permutation. This approach is much faster than the one in [4] because

the representation it uses does not allow gaps or article overlapping, but suffers the restriction that all the articles must be of the same width, while in [4] several article widths are allowed.

3. Proposed approach

In real newspapers, pages are divided into several columns; articles, depending on their length, can occupy one or more columns. Following this idea, the web browser window is divided into several columns of fixed width, fitting in the available surface and avoiding horizontal scroll bars, because it is annoying to scroll the page to the right to finish reading one line and then have to scroll the page to the left to begin reading the following one.

As the number of articles to be returned by the server is unknown a priori, the columns of the page have infinite length, and a vertical scroll bar appears if there is insufficient space for all the articles in the available surface.

All articles to be displayed have a width that is a multiple of the width of one column, depending on their length. The problem to be solved is how to fill all the columns of the web page equally, minimizing gaps between articles. This problem is apparently similar to a two-dimensional bin packing problem [12] if columns are identified with bins. In a bin packing problem, the goal is to minimize the number of bins required to pack several objects, while in this problem, the number of columns (bins) is fixed a priori and what has to be minimized is the unused capacity of the columns.

This representation was first used in [5], but in that approach articles were restricted to fit in only one column, so it was impossible to generate gaps between articles and the fitness function only had to compare the capacity used of the most filled column with the capacity used of the least filled column to estimate the unused capacity. Since in the approach presented in this paper articles may have different widths, it is possible to generate gaps between articles (Figs. 1 and 2), so the fitness function and the greedy algorithm to allocate the articles in the web page have been changed to support this new feature and to minimize the size of the gaps between articles.

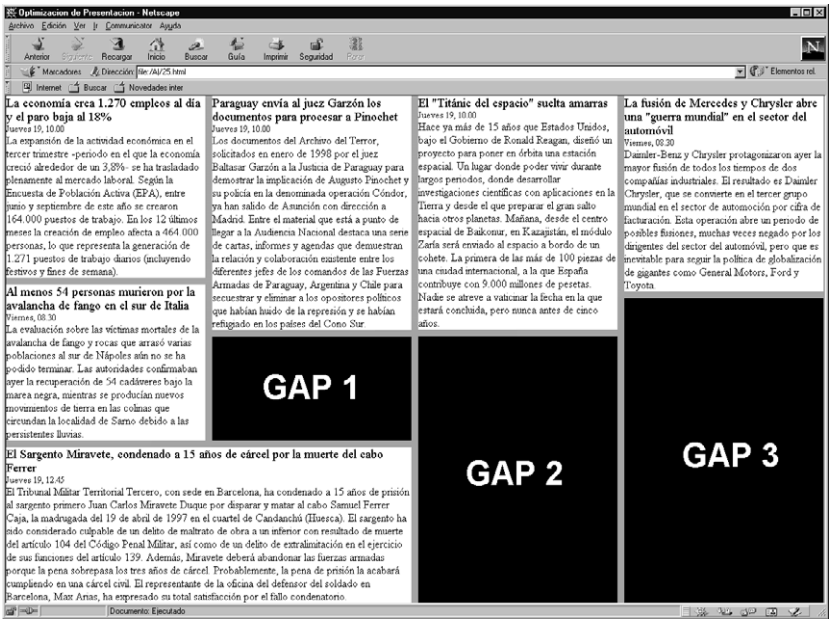

Fig. 1. Example of layout with gaps.

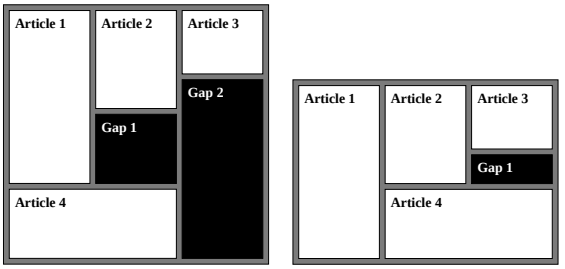

Fig. 2. Bad and good allocation of articles.

SA [1,10] has been widely used to solve combinatorial problems. It is inspired by the physical process of heating a substance and then cooling it slowly until a strong crystalline structure is obtained. This process is simulated by lowering an initial temperature by slow stages until the system "*freezes*" and no more changes occur. Each stage in the process consists in changing the configuration several times until a *thermal equilibrium* is reached and a new stage can start with a lower temperature; the configuration is obtained in the last stage. The changes in the configuration are performed in the following way: a new configuration is built by a random displacement of the current one. If the new configuration is better, then it replaces the current one, and if not, it may replace the current one probabilistically. The probability of the current configuration being replaced by a worse one is high at the beginning of the process and decreases in every stage. This procedure allows the system to move consistently towards the best configuration, yet still enables escape from local optima due to the probabilistic acceptance of worse configurations during the optimization task.

The most important parts of an SA algorithm are: the objective function to be minimized during the optimization process, the representation chosen for potential solutions to the problem and the mutation or configuration change operator. These three characteristics are presented in the next subsections.

3.1. Objective function

The objective function estimates the cost of a layout by measuring the unused surface or number of gaps in it. Gaps may appear in a layout on attempting to allocate an article that is wider than a column as is the case of the gap in the second column in Fig. 1 or when there are no more articles to place and there is unused space in some columns as is the case of gaps in third and fourth columns in Fig. 1.

The objective or cost function to be minimized is the mean height of all the gaps in the layout:

$$C = \frac{\sum_{j=0}^{n} \mathrm{gap}_j}{n}, \tag{1}$$

where n is the number of gaps generated in the layout.

By minimizing this function it is possible to obtain a layout with several gaps, but which are very small, almost imperceptible in most cases.

3.2. Problem representation

Every possible solution has been coded as a permutation of the articles to be displayed. To obtain the layout encoded in the permutation, the system uses a decoder which allocates the articles following the order imposed by the permutation. Each different permutation produces a different layout. With this representation the problem can be seen as a combinatorial problem of size N, with N being the number of articles to be laid out.

The decoder uses a greedy algorithm that allocates all the articles following the order imposed by the permutation encoded in the current configuration. For every article to be placed, it calculates the mean of the gap sizes generated above the article and their dispersion (m_i and d_i) for each column i where it is possible to allocate the article:

$$m_i = \frac{\sum_{j=0}^{n_i} \mathrm{gap}_j}{n_i}, \tag{2}$$

$$d_i = \frac{\sum_{j=0}^{n_i} |\mathrm{gap}_j - m_i|}{n_i}, \tag{3}$$

where n_i is the number of gaps generated above the article when allocating it in column i and gap_j is the height of the jth gap generated.

Finally, the column with the smallest dispersion d_i will be chosen. If there is more than one column with minimal dispersion, the minimal mean m_i is

used to choose the column, and if there is more than one column with equal dispersion and mean values, it chooses the leftmost one (the one with smallest i). The functioning of the decoder is given by the following algorithm:

for $a = 1$ **to** N
  for $i = 1$ **to** $N_c - w_a$
    calculate m_i and d_i following Eqs. (2) and (3)
  endfor
  $SC_1 = \{i: d_i \leqslant d_l;\ i, l = 1, \ldots, N_c - w_a\}$
  $SC_2 = \{i: m_i \leqslant m_l;\ i, l \in SC_1\}$
  $i_{winner} = \min\{i: i \in SC_2\}$
  Allocate article a on column i_{winner}
endfor

where N_c is the maximum number of columns into which the client browser window is divided and w_a is the number of columns taken up by the article a.

Allocating the articles in this way produces layouts where there might be many gaps, but with a very small size, almost imperceptible in most cases. Nevertheless, minimizing the number of gaps as objective function would result in a layout with fewer but bigger gaps, which is worse from the aesthetic point of view (Fig. 2).

3.3. Mutation operator

To obtain a new configuration randomly in the neighborhood of the current one, a transposition of two articles is performed in the following way. For a given configuration, two articles are selected randomly (the underlined ones):

old = (1 2 <u>3</u> 4 5 6 7 <u>8</u> 9).

The new permutation generated is a copy of the old one, but with the numbers at the marked positions swapped:

new = (1 2 <u>8</u> 4 5 6 7 <u>3</u> 9).

3.4. Algorithm

The algorithm that performs the optimization is detailed in the following code:

$v = 0$
$T = T_0$
select a configuration c_{cur} at random
evaluate c_{cur}

repeat
  for $j = 1$ **to** k
    select a new configuration c_{new} in the neighborhood of c_{cur} by mutating c_{cur}
    $\Delta C = C(c_{new}) - C(c_{cur})$
    if ($\Delta C < 0$) OR (random$(0, 1) < e^{-\Delta C/T}$)
    then $c_{cur} = c_{new}$
  endfor
  $v = v + 1$
  $T = f_T(T_0, v)$
until ($T < T_{min}$)

Use the last configuration to obtain the layout where v counts the number of iterations performed, T keeps the current temperature, T_0 is the initial temperature and T_{min} is the minimum temperature to reach, c_{cur} keeps the current configuration, c_{new} is the new configuration selected in the neighborhood of c_{cur}, C is the objective function, k is the number of changes to reach the thermal equilibrium and f_T is the freezer function.

The initial temperature is calculated following Kirkpatrick's suggestion [9]:

$$T_0 = -\frac{\Delta C^*}{\ln(p_a)} \tag{4}$$

where ΔC^* is the average objective increase observed in a random change, and p_a is the initial acceptance probability (0.8 is usually used).

For the freezer function (f_T) this approach uses

$$f_T(T_0, v) = \frac{T_0}{1 + v}. \tag{5}$$

This function lowers the temperature and thus the acceptance probability of a solution is worse than the current one. Initially the temperature is high, producing an exploration of the search space, followed by a controlled descent where the algorithm searches locally until the minimum temperature is reached.

The minimum temperature is calculated on the basis of the desired number of iterations *numIt* as follows

$$T_{min} = f_T(T_0, numIt). \tag{6}$$

4. Algorithm parameters

The behavior of the above algorithm is determined by the parameters k (number of changes to reach the thermal equilibrium) and *numIt* (number of iterations). The values of the above parameters have a strong influence on the solutions found by the proposed algorithm. These parameters influence both the cost achieved and the time or computational complexity required.

To determine heuristic values for k and *numIt*, an experiment has been designed consisting of a set of 14 articles for which, at least one optimum presentation is known (see Fig. 3). Parameters k and *numIt* are then varied to study their influence on the total execution time and the quality of the results obtained. Parameter k was assigned values of {1, 5, 10, 15, 20, 25, 30} and parameter *numIt* was assigned values of {10, 20, 30, 40, 50, 60, 70, 80}. For each combination of parameters, the algorithm was executed 10 times to obtain a mean value and standard deviation of the time spent to execute the algorithm and the cost of the solution found.

As shown in Fig. 4(a), execution time (t) has a linear dependency on both k and *numIt*, i.e., a dependency of the type $O(p \cdot k \cdot numIt)$, where p is a constant that depends on the size of the problem.

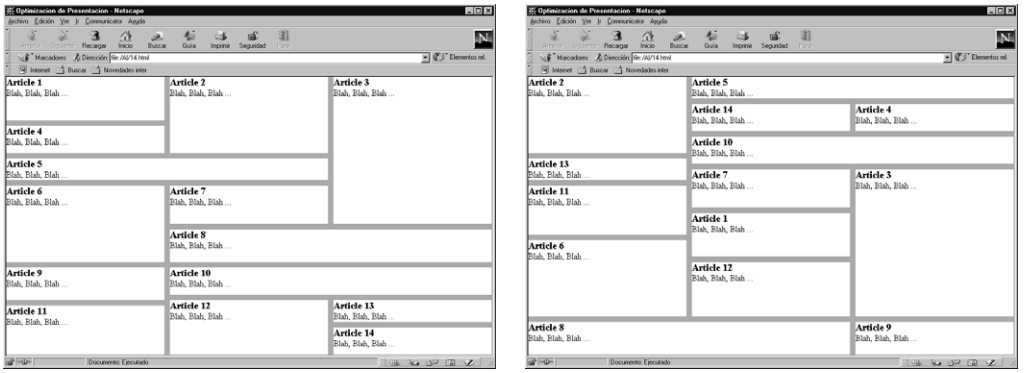

Fig. 3. (a) Optimum presentation, determined a priori, for a problem of 14 articles. (b) One of the optimum presentations found by the algorithm.

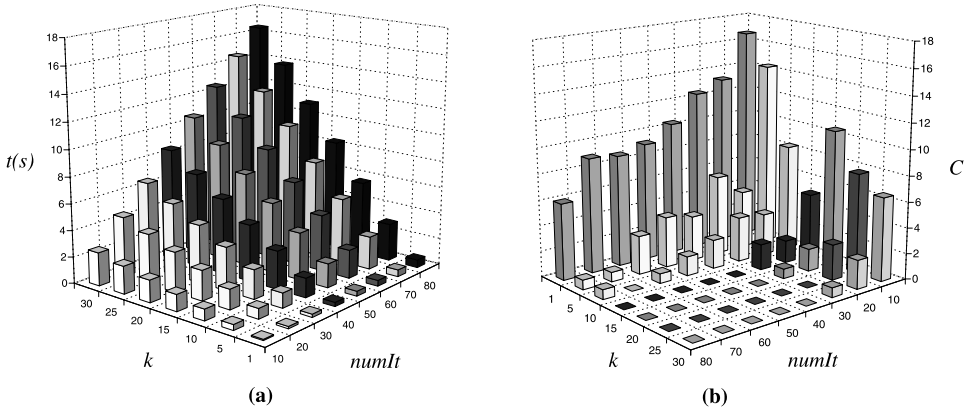

Fig. 4. Mean execution time in seconds (a) and mean cost in pixels (b) obtained for a problem of 14 articles with 10 runs for each combination of parameters.

Another relevant aspect is that, as shown in Fig. 4(a), the parameter k has a more direct influence on the execution time than *numIt*. Therefore, we wish to set a small value for the parameter k and a larger one for *numIt* in order to minimize the time spent in the optimization process.

Concerning the quality of the results obtained, Fig. 4(b) shows that from $k = 15$ with a sufficient number of iterations ($numIt \geqslant 40$), the optimum presentation is found for the problem in each of the 10 executions for each combination of parameters. Table 1 shows the percentage of optimum presentations found among all the simulations performed.

From the above results, when the number of iterations is not too small, the thermal equilibrium seems to be achieved when $k \geqslant 15$. To ensure that this is a good value for k, we designed a further experiment, equal to the previous one but with 30 articles. The parameter k was adjusted to values around 15. The results obtained are shown in Fig. 5.

Again, with a number of iterations that is appropriate for the complexity of the problem ($numIt \geqslant 200$) and with a value of k greater than or equal to 15, an optimum presentation is obtained in all the executions of the algorithm. These two experiments reveal that when k has a value around 15 thermal equilibrium is reached and the algorithm presents a fairly robust behavior. The number of iterations, however, is totally dependent on the complexity of the problem.

Another important question is whether it is necessary to find always the optimum presentation. For the problem presented in this paper, the answer is no. The aim is to find a good presentation as soon as possible, as the system must function in real time. On most occasions, finding the optimum solution requires too much time, while a good solution (with a cost of less than 10–15 pixels) can be obtained much faster, thus meeting the real-time restrictions established a priori.

Taking into account that the maximum number of articles returned by the server at the client's request can be set to 25 (to avoid information saturation), and that real-time restrictions must be met, the parameter *numIt* is set at 80, which

Table 1
Percentage of optimum presentations found for a problem with 14 articles

k	*numIt* (%)							
	10	20	30	40	50	60	70	80
1	0	0	0	30	20	20	10	40
5	10	50	40	60	60	60	90	90
10	30	60	70	70	80	90	100	90
15	60	80	80	100	100	100	100	100
20	20	80	80	100	100	100	100	100
25	40	70	100	100	100	100	100	100
30	50	70	90	100	100	100	100	100

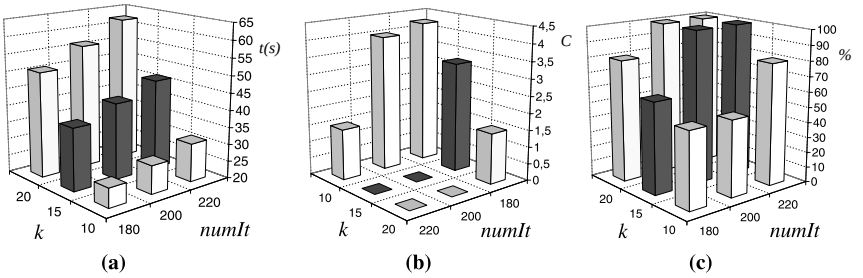

Fig. 5. Mean execution time in seconds (a), mean cost in pixels (b) and percentage of optimum presentations found (c) for a problem of 30 articles with 10 runs for each combination of parameters.

implies an execution time of 10–15 s for 25 articles, depending on the speed of the computer it is run on.

5. Results

The algorithm has been tested with several articles extracted from the Spanish web newspaper **EL MUNDO** (http://www.elmundo.es). Articles have been extracted manually from the web newspaper. In the future will be added a module to extract them automatically following some user guidelines.

Table 2 shows the minimum, maximum, average and standard deviation of execution time (t) in seconds and cost (unused surface) (C) in pixels measured over 10 runs for each number of articles executed within Netscape Communicator 4.5 running in a 233 MHz Intel Pentium II. The parameters used in all the algorithm runs were $numIt = 80$ and $k = 15$.

One interesting feature of this algorithm is that the average search time increases linearly with the size of the problem, as shown in Fig. 6, where the minimum, average and maximum times in Table 2 are plotted.

The algorithm is capable of finding very good solutions in a reasonable time with only 80 iterations. Although a maximum of 25 articles is returned by the server, as a higher number would produce too much information for a single web page, Table 2 shows that even if the number of articles were greater, the algorithm would still be capable of finding a good solution in real time.

Since the algorithm is written in *JavaScript* [2], it must be interpreted within the web browser, which is a slower process than running a compiled program. Even under this restriction, it obtains

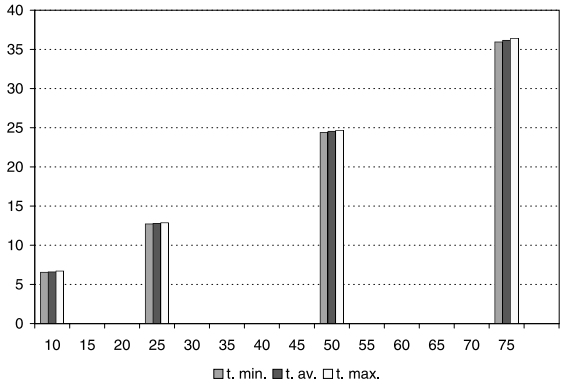

Fig. 6. Minimum, average and maximum time in optimizing 10, 25, 50 and 75 articles.

reasonable times compared with the time spent while the web page is loaded, and taking into account that in a normal-sized browser window with a standard (readable) font size, i.e., 10–12 points, there is only room for 10 articles without scrolling bars in the window and that the maximum number of articles returned from the server is 25, optimization times are usually between 6.6 s (with 10 articles) and 12.8 s (with 25 articles), which is acceptable for a personalized optimization of the layout.

An example of a final result is shown in Fig. 7, where 25 articles are displayed using a very small font size to obtain a layout that fits inside a window without scrolling bars.

6. Future work

One module that is necessary for the functioning of the system is the automatic extraction of stories from the newspaper web page according to

Table 2
Minimum, maximum, average and standard deviation of time and cost (mean of gap sizes in the layout) optimizing 10, 25, 50 and 75 articles

Articles	t_{min}	t_{max}	$t \pm \sigma$	C_{min}	C_{max}	$C \pm \sigma$
10	6.55	6.70	6.60 ± 0.05	2.00	2.67	2.47 ± 0.32
25	12.72	12.87	12.78 ± 0.04	0.67	3.40	2.34 ± 0.77
50	24.40	24.65	24.54 ± 0.07	2.00	9.14	4.61 ± 2.00
75	35.94	36.40	36.16 ± 0.15	3.75	10.75	6.82 ± 2.35

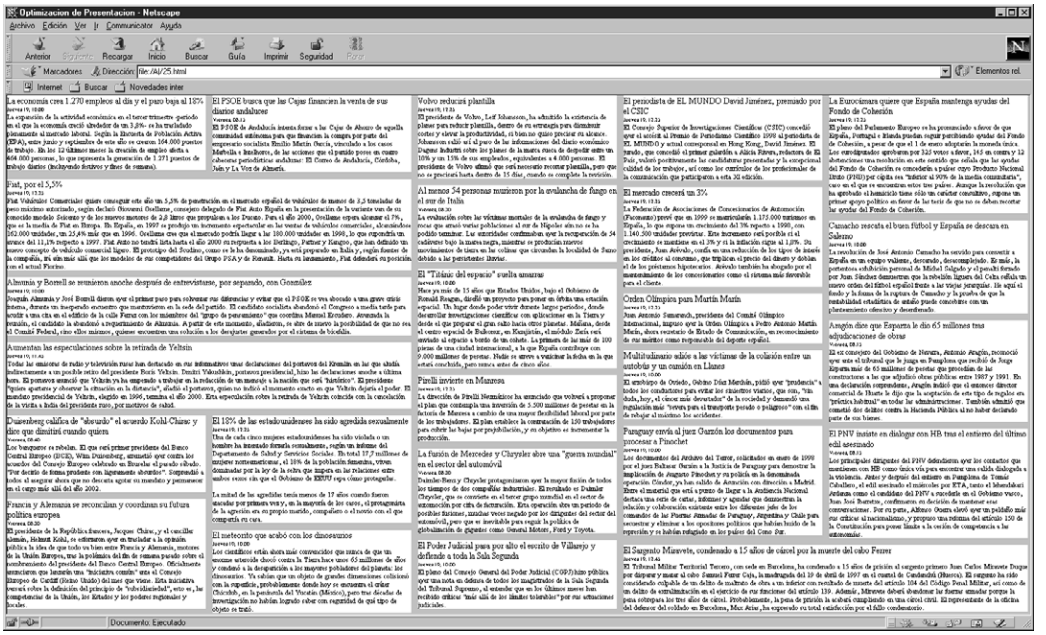

Fig. 7. Final result of a simulated newspaper page with 25 articles.

the user's criteria. As each newspaper uses a different format to display its articles, and as the designers of the newspaper pages normally use visual editors to generate them, the news extractor must take into account many possibilities in order to detect all of the stories. This module is currently being developed, though we hope to have a beta version functioning soon. Such a module simply looks for coincidences of key words within the text of the articles, giving greater weight to the article if the word is found within the title. Once all the stories containing a key word have been extracted, they are classified by the number of agreements and their relevance; they are then sent in groups of 25 articles, starting with the most relevant.

Another desirable characteristic for the automatic paginator would be for it to place in proximity stories that deal with similar topics, in order to facilitate overall comprehension for the user. Implementation of this second characteristic requires each article sent by the server to be studied, and so this might not be included in the real-time system, though it would be of great interest for off-line requests.

7. Conclusions

This paper presents a new, SA-based approach to the web newspaper layout problem. It differs from [5] in that it adds the feature of being able to deal with articles with different widths. As in [5], the optimization program is a script written in *JavaScript* embedded in the web page to be optimized, which will be executed in the client's machine, adapting itself to the client web browser configuration.

The time required for the optimization process is acceptable compared with the usual time the user has to wait while the web page loads; for example, in a 233 MHz Intel Pentium II it is usually between 6.6 s (with 10 articles) and 12.8 s (with 25 articles), and with current processors the optimization time should be better, which is very acceptable taking into account that the optimization task is executed by the *JavaScript* engine within the browser.

The proposed approach is available at http://atc.ugr.es/~jesus/layout for anyone interested in executing or taking a look at the algorithm.

References

[1] E.H.L. Aarts, J. Korst, in: Simulated Annealing and Boltzmann Machines, Willey, New York, 1989.

[2] Netscape Communications Corporation, JavaScript Developer Central, Web address: http://developer.netscape.com/tech/javascript.

[3] D.E. Goldberg, in: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[4] J. González, J.J. Merelo, Optimizing web page layout using an annealed genetic algorithm as client-side script, in: Lecture Notes in Computer Science, vol. 1498, Springer, New York, 1998, pp. 1018–1027.

[5] J. González, J.J. Merelo, P.A. Castillo, V. Rivas, G. Romero, Optimizing web newspaper layout using simulated annealing, in: Lecture Notes in Computer Science, vol. 1607, Springer, New York, 1999, pp. 759–768.

[6] W.H. Graf, Graf's home page, Web address: http://www.dfki.de/~graf/.

[7] J.J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[8] O. Kamba, K. Bharat, M.C. Albers, The Krakatoa chronicle – an interactive, personalized newspaper on the Web, Technical Report Number 95-25, Technical Report, Graphics, Visualisation and Usability Center, Georgia Institute of Technology, Atlanta, GA, 1995.

[9] S. Kirkpatrick, Optimization by simulated annealing – quantitative studies, J. Stat. Phys. 34 (1984) 975–986.

[10] S. Kirkpatrick, C.D. Gerlatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[11] K. Lagus, I. Karanta, J. Ylāa-Jāāski, Paginating the generalized newspapers – a comparison of simulated annealing and a heuristic method, in: H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.), Proceedings of the Fourth Conference on Parallel ProblemSolving from Nature, Lecture Notes in Computer Science, vol. 1141, Springer, Berlin, September 1996, pp. 595–603.

[12] S. Martello, P. Toth, in: Bin Packing Problem, in: Knapsack Problems, Algorithms and Computer Implementations, Wiley, New York, 1990 (Chapter 8).

[13] Z. Michalewicz, in: Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer, New York, 1996.

Jesús González was born in 1974. He received the M.Sc. degree in Computer Science in 1997 from the University of Granada, Spain. He is currently a Fellowship holder and Doctorate Student within the Department of Computer Architecture and Computer Technology at the same University. His current areas of research interest are in the fields of function approximation using radial basis function neural networks, fuzzy systems, and genetic algorithms.

Ignacio Rojas received his M.S. in Physics and Electronics from the University of Granada in 1992 and his Ph.D. degree, in the field of Neuro-Fuzzy in 1996. During 1998 he was visiting professor of the BISC at the University of California, Berkeley. He is currently an Associate Professor at Department of Computer Architecture and Computer Technology. His main research interests are in the fields of hybrid system and combination of fuzzy logic, genetic algorithms and neural networks, financial forecasting and financial analysis.

Hector Pomares received his M.Sc. degree in Electronics Engineering in 1995, the M.Sc. degree in Physics in 1997, and the Ph.D. degree in 2000, all from the University of Granada, Granada, Spain. At present, he is an Associate Professor at the Department of Computer Architecture and Computer Technology. His current areas of research interest are in the fields of fuzzy approximation, neural networks and genetic algorithms, and Radial Basis Function design.

Moisés Salmerón received his B.Sc. degree in 1994 and his M.Sc. degree in 1997, both in Computer Science by the University of Granada, Spain. His last-year engineering project was about the design of testable neural networks. In September, 1997 he joined the CASIP (Circuits and Systems for Information Processing) research group at the Department of Computer Architecture and Computer Technology, where he is a researcher working towards his Ph.D. dissertation which deals with neural networks for time series prediction. His current research interests are in the fields of neural networks, modern control theory, time series statistical modelling, and application of matricial techniques such as SVD, PCA and orthogonal transformations for the design of neural networks and hybrid systems for time series prediction.

Alberto Prieto received a B. Sc. degree in Electronic Physics in 1968 from the Complutense University (Madrid) and Ph.D. degree from the University of Granada, Spain, in 1976. From 1971 to 1984 he was Director of the Computer Centre and from 1985 to 1990 Dean of the Computer Science and Technology studies of the University of Granada. In June to August 1991, he was at the T.I.R.F. Laboratory, Institute National Polythechnique of Grenoble, as guest researcher of Profs. J. Herault and C. Jutten. During 1991–1992, jointly with Prof. P. Treleaven, he was main researcher of a Spanish–British Integrated Action on Artificial Neural Networks, held between University of Granada and the University College of London. He is currently a Full Professor in the

Department of Computer Architecture and Computer Technology. His research interests are in the areas of artificial neural networks, fuzzy systems, expert systems and microprocessor-based systems. He is a member of INNS, AEIA, AFCET, IEEE and ACM, and Chairman of the Spanish Regional Interest Group of the IEEE Neural Network Council.

J.J. Merelo was born in Ubeda, Jaén, Spain, on March 10, 1965. Received a degree in Theoretical Physics and a Ph.D. in Physics by the University of Granada in 1988 and 1994. He is currently an Associate Professor at the University of Granada, attached to the Department of Computer Architecture and Computer Technology. His main areas of research are internet computing, neural nets, and evolutionary computation.