Hybrid Genetic Algorithms for Scheduling Advertisements on a Web Page

Subodha Kumar
University of Washington

Varghese Jacob
University of Texas at Dallas

Chelliah Sriskandaraj
University of Texas at Dallas

HYBRID GENETIC ALGORITHMS FOR SCHEDULING ADVERTISEMENTS ON A WEB PAGE

Subodha Kumar
University of Washington
subodha@u.washington.edu

Varghese S. Jacob
University of Texas at Dallas
vjacob@utdallas.edu

Chelliah Sriskandarajah
University of Texas at Dallas
chelliah@utdallas.edu

Abstract

Many web sites provide free services to users. The revenue for these sites is dependent on the advertisements they place on their web pages. For these firms, therefore, the optimal placing of advertisements on their web pages becomes a critical issue. In a given planning horizon, we assume that the number of advertisements available for placement on a web site is greater than the available space. The site, therefore, needs to determine the optimal allocation of advertisement space. We develop a hybrid genetic algorithm (GA) that uses problem specific knowledge during the evolution of solutions to solve this problem. Our initial computational results show that the hybrid GA performs exceptionally well in the sense that it provides optimal or near optimal solutions for a variety of problems.

Keywords: Hybrid genetic algorithm, scheduling advertisements, web page.

RESEARCH OBJECTIVES AND QUESTIONS

Banner ads are one of the popular forms of web advertising (Rewick 2001). Typically, a banner ad is a small graphic image that is linked to a target ad (Novak and Hoffman 1997). Rectangular-shaped banner ads are the most common type of banner ads. These banners usually appear on the side, top, or bottom of a screen as a distinct, clickable image (McCandless 1998).

The banner ad display on a web page is specified in the form of a rectangular slot of width W and height S. The slot can appear on a screen in any format. One such format, shown in Figure 1, is a screen where one slot of side banner ads is displayed. This slot has three different ads, namely A_2, A_3, and A_5. All of these ads have the same width W and their heights are s_2, s_3, and s_5 respectively. The ads in this slot are displayed to the users accessing the web page in a particular time interval.

We observe various types of banner ads on different sites. For example, *www.cheaptickets.com* (accessed on September 5, 2001) displays three side banner ads and *www.travelocity.com* (accessed on September 5, 2001) displays six side banner ads of different heights. Typically, a set of ads competes for space on web page in a planning horizon (say a day). A day consists of 24 x 60 minutes. If ads are updated every 20 seconds on the web page, we refer to each 20 seconds as a time interval. Thus, in this case we have 3 x 24 x 60 time intervals to schedule ads. Usually a site displays ads in consecutive intervals of a planning horizon and sells ad space to different sponsors in each planning horizon. In each time interval, a rectangular slot (e.g., side banner) consisting of ads is displayed for viewers. It was observed that “UtopiAd's Magellan” (accessed on September 5, 2001) updated their ads every 20 seconds.

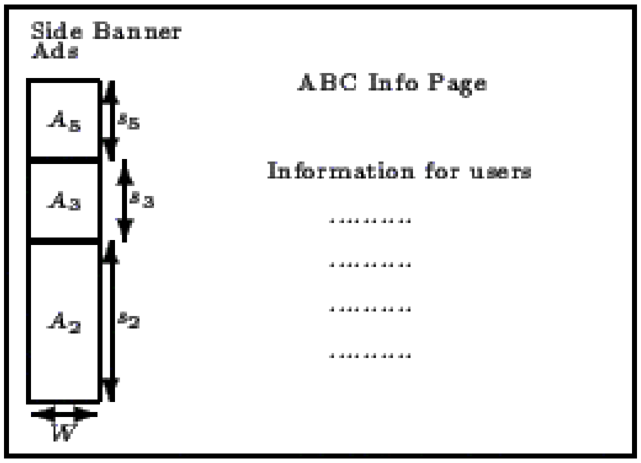


Figure 1. Screen Showing the Format of a Slot

In general, in a given planning horizon (time frame), there are a fixed number of slots available and only a subset of ads competing for the space can be placed in these slots. This problem is called the MAXSPACE problem (Dawande et al. 2001); it is explained in the following section.

THEORETICAL FOUNDATIONS OF THE STUDY

In this section, we provide notations and a mathematical formulation of the problem. We consider a set of n ads $A = \{A_1, ..., A_n\}$ that are competing for space in a given planning horizon, which is divided into N time intervals. The space allocated for ads on a web page is in the form of rectangular slot of size: S x W, where S is the height and W is the width. The set of ads assigned to a slot is shown in a time interval to the visitors who visit the site during that time interval. Ad A_i has *height* s_i and *frequency* w_i. The width of all ads is W. All of the selected ads are to be placed in N slots, where each slot has height S. For an ad A_i, its height s_i represents the amount of space the ad occupies in a slot while its frequency w_i represents the number of slots in which the ad must appear. In other words, w_i copies of ad A_i must appear in the slots. Clearly, $w_i \leq N \; \forall \; i$. Also, advertisers do not want to display an ad in the same slot more than once. Thus, ad A_i can be displayed at most once in a slot. Ad A_i is said to be *scheduled* if exactly w_i copies of A_i appear in the slots and each slot contains at most one copy of the ad. The total height of the ads placed in a slot cannot exceed S. A feasible schedule for this problem is a placement of a subset $A' \subseteq A$ of ads such that the following conditions are satisfied: (1) for each $A_i \in A'$, exactly w_i copies are placed in the slots with at most one copy in each slot and (2) for $j = 1, ..., N$, the sum of ad sizes assigned to slot j must not exceed S. That is, $\sum_{A_i \in B_j} s_i \leq S, \forall \; j$, where $B_j \subseteq A'$ is a set of ads which have a copy in slot j. Clearly, $s_i \leq S$, $\forall \; i$.

The objective of the MAXSPACE problem is to find a feasible schedule of $A' \subseteq A$ ads such that the total *weight* $\sum_{A_i \in A'} s_i w_i$ is maximized. A feasible schedule for the problem instance in Figure 2(a) for $S = 8$ is shown in Figure 2(b) with $\sum_{A_i \in A'} s_i w_i$ = 34, where $A' = \{A_1, A_3, A_4, A_5, A_6, A_7\}$. An optimal schedule is given in Figure 2(c) with $\sum_{A_i \in A'} s_i w_i$ = 40, where $A' = \{A_1, A_2, A_3, A_4, A_5, A_7\}$. For a given schedule, the *fullness* of slot j is defined as $f_j = \sum_{A_i \in B_j} s_i$, where B_j is a set of ads that have a copy in slot j. For example, a feasible schedule for the problem instance given in Figure 2(a) is shown in Figure 2(b) and the fullness of slot 3 is $s_4 + s_5 = 8$. Maximum fullness of the slots is defined as $max_j f_j$. For the feasible schedule in Figure 2(b), maximum fullness, f = $\max_j f_j = f_1 = 8$.

For both the schedule shown in Figures 2(b) and 2(c), it should be noted that at any time interval only one of the five slots is shown to a particular user accessing the web page. For example, side banner ads displayed in Figure 1 correspond to slot 5 of

Figure 2(b). Note that the choice of which slot to display in a time interval can be made by cycling through a deterministic permutation of *N* slots.

Adler et al. (2001) provide a heuristic algorithm (called SUBSET-LSLF) for the MAXSPACE problem. Kumar et al. (2001) developed a hybrid genetic algorithm (GA) integrating SUBSET-LSLF with GA.

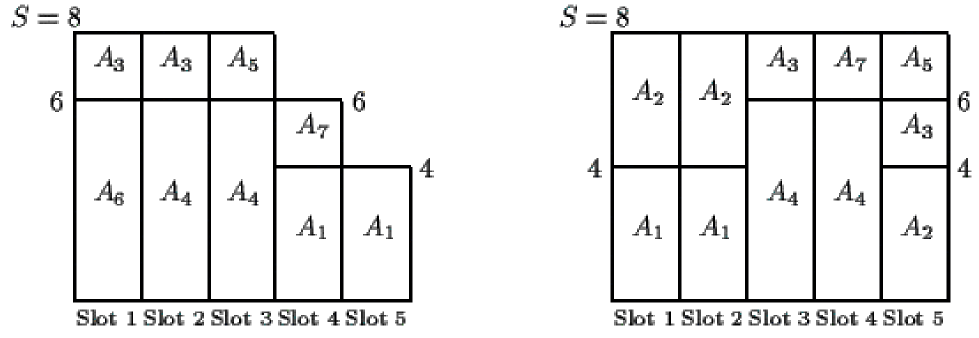
RESEARCH METHODOLOGY BEING USED

Since the MAXSPACE problem is NP-Hard (Dawande et al. 2001), it is unlikely that the problem can be solved by an efficient optimal algorithm (Garey and Johnson 1979) and we need to develop heuristic algorithms.

We select the ads from the set of available ads and schedule them in such a way that the space utilization is maximized. The schedule provides details of the identity of ads that appear together and the time in which interval they appear.

Ad A_i	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
s_i	4	4	2	6	2	6	2	5	5
w_i	2	3	2	2	1	1	1	1	1

(a) Problem data


$S = 8$
A_3 A_3 A_5
6
A_7
6
4
A_6 A_4 A_4 A_1 A_1
Slot 1 Slot 2 Slot 3 Slot 4 Slot 5
$S = 8$
A_2 A_2 A_3 A_7 A_5
6
A_3
4
4
A_1 A_1 A_4 A_4 A_2
Slot 1 Slot 2 Slot 3 Slot 4 Slot 5

(b) A feasible, nonoptimal schedule (c) An optimal schedule

Figure 2. An Example to Illustrate MAXSPACE Problem

Integer Programming Formulation

$$\max \left\{ \sum_{j=1}^{N} \sum_{i=1}^{n} s_i x_{ij} : \quad \sum_{i=1}^{n} s_i x_{ij} \leq S, \ \ j = 1,2,\ldots\ldots,N; \quad \sum_{j=1}^{N} x_{ij} = w_i y_i, \ i = 1,2,\ldots\ldots,n \right\}$$

where $x_{ij} = \begin{cases} 1 & \text{if ad } A_i \text{ is assigned to slot } j \\ 0 & \text{otherwise.} \end{cases}$; $y_i = \begin{cases} 1 & \text{if ad } A_i \text{ is assigned} \\ 0 & \text{otherwise.} \end{cases}$

We used the *CPLEX Linear Optimizer 6.0.1* to solve the problem. CPLEX fails to find the optimal solution for large size problems with $N \geq 25$ as the memory space is exceeded. However, CPLEX provides an upper bound for all the problems. The performances of the heuristic methods are evaluated by comparing with these upper bounds.

LSMF

We develop an algorithm, called LSMF, in which we first determine the maximum slot fullness. If the maximum slot fullness obtained is less than or equal to the available space for each slot (S), then we have a feasible solution, otherwise we discard some of the ads.

Since the MAXSPACE problem belongs to the class of packing problems, we develop a procedure called SUBSET-LSMF based on the idea of the Multifit algorithm for the classical bin packing problem (Coffman et al. 1978). SUBSET-LSMF finds the minimum slot size required. It sets a lower and an upper bound on sizes of all the slots. It sorts the ads by size (s_i) in non-increasing order. Then using a procedure called "First Fit Decreasing" (FFD), it determines the number of slots. FFD assigns the ads in the sorted order such that the w_i copies of ad A_i is assigned to the w_i most full slots with no more than one copy of the same ad in any slot. If the determined number of slots is greater than the given number of slots N, it increases the lower bound on sizes of all the slots, otherwise the upper bound is reduced. This process is repeated until the lower and upper bound converges to a very small interval and a feasible schedule is found.

Algorithm LSMF

```
Step 1.  Run SUBSET-LSMF and determine the maximum fullness of slots, $f$.
Step 2.  If $f \leq S$, then terminate, otherwise go to step 3.
Step 3.  Let $B_i = s_i w_i$ for ad $i$. Calculate $B_i$ for all ads $i = 1, 2, \ldots, n$.
Step 4.  Sort the ads by $B_i$, from smallest to largest.
Step 5.  Set $k = 1$. Remove the first sorted ad from the schedule and run SUBSET-LSMF again to determine the value of $f$. Store
         the removed ad as Discard ($k$).
Step 6.  If $f \leq S$, go to step 8, otherwise go to step 7.
Step 7.  Set $k = k + 1$. Remove the next sorted ad from the schedule and run SUBSET-LSMF again to determine the value of $f$.
         Store the removed ad as Discard ($k$). Go to step 6.
Step 8.  If $k = 1$, then terminate, else add the ad Discard($k$ - 1) in the schedule and run SUBSET-LSMF again to determine the
         value of $f$.
Step 9.  If $f > S$, go to step 10, otherwise go to step 11.
Step 10. Remove the ad Discard ($k$ - 1) from the schedule.
Step 11. Set $k = k - 1$. Go to step 8.
```

At step 1, LSMF determines the maximum slot fullness. If this maximum slot fullness is less than or equal to S, it stops at step 2. If the maximum slot fullness is greater than S, the ads with the smallest values of $s_{i.} w_i$ are discarded until the maximum slot fullness becomes less than or equal to S. Once the maximum slot fullness is less than or equal to S, some of the discarded ads are again added such that the maximum slot fullness remains less than or equal to S.

A Genetic Algorithm

Kumar et al. (2001) develop a hybrid GA by combining GA with the SUBSET-LSLF approach (Adler et al. 2001). However, a preliminary analysis indicates that the LSMF approach dominates the SUBSET-LSLF approach. In this research, therefore, we develop a hybrid GA-LSMF algorithm for solving the problem and evaluate its performance. We use "one-point" crossover and the arbitrary two-ad change mutation.

Algorithm GA-LSMF

```
Step 1.  Initialize population size ($ps$), crossover probability ($p_c$), mutation probability ($p_m$), elite fraction ($\varepsilon$) and number of
         generations ($n_{gen}$). Set $i = 0$.
Step 2.  Generate $ps$ - 1 random sequences and one sequence using algorithm LSMF.
Step 3.  Assign ads as per the assignment in algorithm LSMF.
Step 4.  Fitness value for each sequence is evaluated based on the maximum space utilization (i.e., $\sum_{i \in A'} |s_i w_i|$) for that string.
         Here $A'$ is the subset of ads having all ads that can be placed without violating the size limit of any slot.
Step 5.  Sort all the sequences in descending order of their fitness values.
Step 6.  Select $\varepsilon$ upper fraction of the population and reproduce them in proportion of their relative fitness.
Step 7.  Set $k = 0$.
Step 8.  Select two parents from the population randomly according to their relative fitness and crossover to obtain two children.
         Set $k = k + 1$.
Step 9.  Mutate the children based on their mutation probabilities.
Step 10. Estimate the fitness value of the children using steps 3 and 4.
```

Step 11. If k <((ps/2) - 0.5), go to Step 8.
Step 12. If $i = 0$, set the overall best sequence = current best sequence and go to step 14.
Step 13. If the overall best sequence is better than the current best sequence, replace the current worst sequence with the overall best sequence; else, set the overall best sequence = current best sequence.
Step 14. Set $i = i + 1$. If $i = n_{gen}$, terminate; else, go to Step 5.

CURRENT STATUS OF THE PROJECT

For computational studies, we use 150 randomly generated problems. Problems are generated in such a way that $\sum_{i \in A} | s_i w_i | = N * S$ for any test problem and all the ads fit into the given N slots. Here, A is the set of ads generated for the test problem. Therefore, the optimal values are known and for any test problem it is equal to N * S. These problems are generated for 15 different combinations of N and S.

The size s_i of ad A_i in any test problem is generated randomly between $S/3$ and $2S/3$, where S is the size of each slot for that problem. It is found that the problems generated with these limits on the value of s_i are more difficult for the proposed heuristic to solve than randomly generated problems without any limits on the value of s_i.

A statistical experimental framework is used to discover the most appropriate values of the GA parameters. The experimental layout is full-factorial design (Montgomery 1996). This experimental layout used three levels (possible values) each of ε, p_c and p_m. For small size problems, two levels of *ps* are used and for larger size problems, three levels of *ps* are used.

Comparison of Results

In order to evaluate their relative performance, SUBSET-LSLF, LSMF, and the GA are coded in C and executed on a SunOS 5.6 system. The solutions produced by these algorithms are compared with the optimal solution.

We find that CPLEX fails to find the optimal solution for problems with $N \geq 25$ as the memory space is exceeded. CPLEX gives upper bounds for all of the problems. In order to evaluate the performance of algorithms for the test problems with unknown optimal values, we will use the upper bounds obtained by CPLEX.

First the pure heuristics without any hybridization are compared. Table 1 shows the results. Each row in this table is the result for a different combination of N and S and there are 15 rows. For all the test problems, the values of N and S are shown in the table. Percentage SUBSET-LSLF gap in the table is the percentage deviation of SUBSET-LSLF from the optimal solution. It is calculated as follows:

$$Percentage\ SUBSET - LSLF\ Gap = \frac{(Optimal\ Solution) - (SUBSET - LSLF)}{Optimal\ Solution} \times 100\%$$

We have generated 10 problems for each combination of N and S. So, Max, Avg and Min columns show the maximum, average and minimum values of percentage gaps respectively, out of these 10 problems in the set. Similarly, GA percentage gap is the percentage deviation of GA from the optimal solution and LSMF gap is the percentage deviation of LSMF from the optimal solution. We see that the average percentage GA gap is 0% for small size problems ($N = 10$), which means that the GA achieves the optimal solution for all these problems. For medium and large size problems, this average percentage gap is in the range of 0.2% to 2.3%. Table 1 shows that the average percentage LSMF gap is very small for all the problems (in the range of 0% to 1.75%).

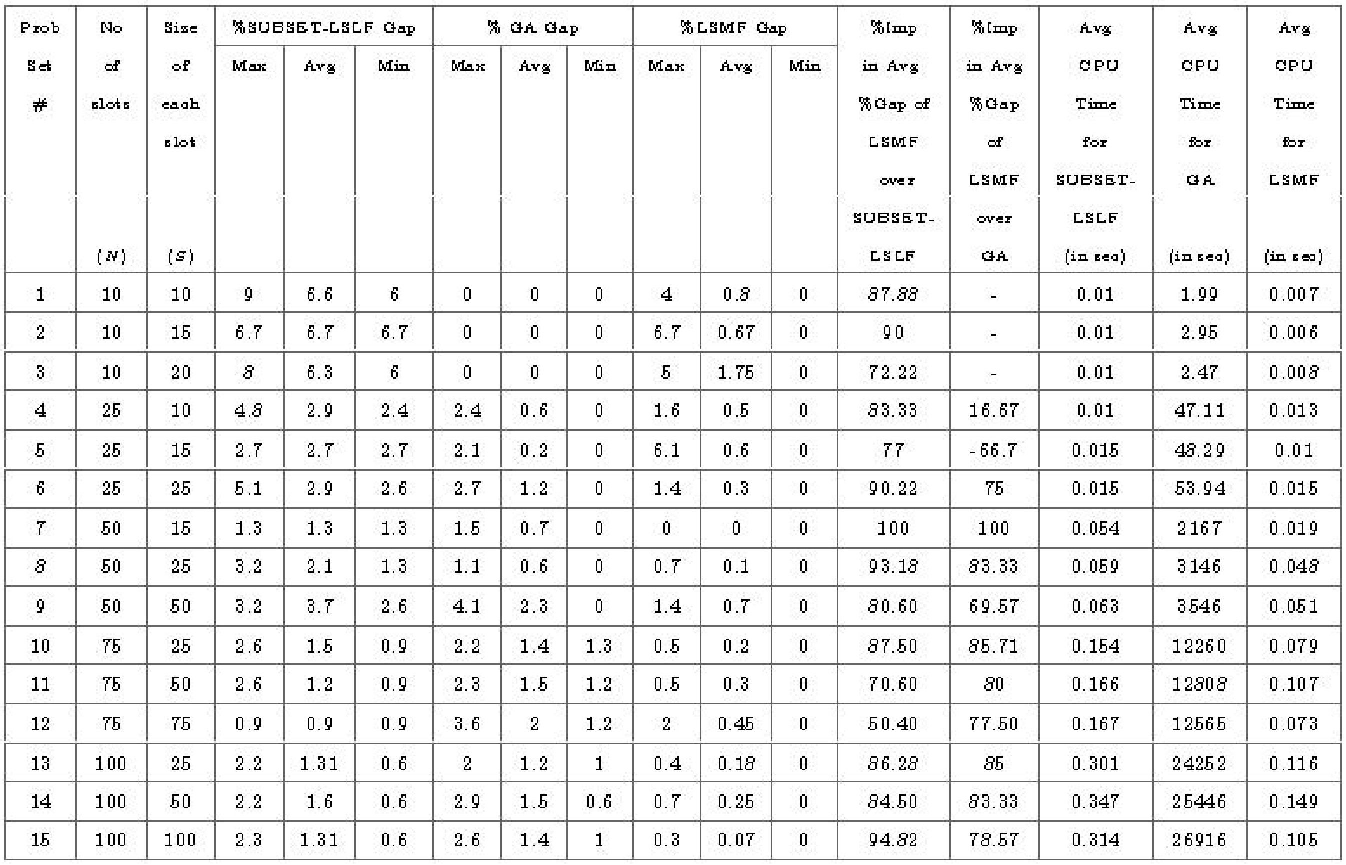
Percentage improvement in average percentage gap of LSMF over SUBSET-LSLF is calculated as follows:

$$Percentage\ Improvement\ in\ Avg\ \%Gap = \frac{(Avg\ \%SUBSET - LSLF\ Gap) - (Avg\ \%\ LSMF}{(Avg\ \%SUBSET - LSLF\ Gap)} \times 100\%$$

We find that the LSMF outperforms SUBSET-LSLF for all the test problems and the improvement is in the range of 50% to 100%.

Percentage improvement in average percentage gap of LSMF over GA is calculated similarly. We find that the LSMF outperforms GA for all the medium and large size test problems (N = 50, 75, and 100) and improvement is in the range of 67% to 100%. For some of the small size problems, GA performs better than LSMF.

Table 1. Comparison of Test Problems Results



Prob Set #	No of slots (*N*)	Size of each slot (*S*)	%SUBSET-LSLF Gap			% GA Gap			%LSMF Gap			%Imp in Avg %Gap of LSMF over SUBSET-LSLF	%Imp in Avg %Gap of LSMF over GA	Avg CPU Time for SUBSET-LSLF (in sec)	Avg CPU Time for GA (in sec)	Avg CPU Time for LSMF (in sec)
			Max	Avg	Min	Max	Avg	Min	Max	Avg	Min					
1	10	10	9	6.6	6	0	0	0	4	0.8	0	87.88	-	0.01	1.99	0.007
2	10	15	6.7	6.7	6.7	0	0	0	6.7	0.67	0	90	-	0.01	2.95	0.006
3	10	20	8	6.3	6	0	0	0	5	1.75	0	72.22	-	0.01	2.47	0.008
4	25	10	4.8	2.9	2.4	2.4	0.6	0	1.6	0.5	0	83.33	16.67	0.01	47.11	0.013
5	25	15	2.7	2.7	2.7	2.1	0.2	0	6.1	0.6	0	77	-66.7	0.015	48.29	0.01
6	25	25	5.1	2.9	2.6	2.7	1.2	0	1.4	0.3	0	90.22	75	0.015	53.94	0.015
7	50	15	1.3	1.3	1.3	1.5	0.7	0	0	0	0	100	100	0.054	2167	0.019
8	50	25	3.2	2.1	1.3	1.1	0.6	0	0.7	0.1	0	93.18	83.33	0.059	3146	0.048
9	50	50	3.2	3.7	2.6	4.1	2.3	0	1.4	0.7	0	80.60	69.57	0.063	3546	0.051
10	75	25	2.6	1.5	0.9	2.2	1.4	1.3	0.5	0.2	0	87.50	85.71	0.154	12260	0.079
11	75	50	2.6	1.2	0.9	2.3	1.5	1.2	0.5	0.3	0	70.60	80	0.166	12808	0.107
12	75	75	0.9	0.9	0.9	3.6	2	1.2	2	0.45	0	50.40	77.50	0.167	12565	0.073
13	100	25	2.2	1.31	0.6	2	1.2	1	0.4	0.18	0	86.28	85	0.301	24252	0.116
14	100	50	2.2	1.6	0.6	2.9	1.5	0.6	0.7	0.25	0	84.50	83.33	0.347	25446	0.149
15	100	100	2.3	1.31	0.6	2.6	1.4	1	0.3	0.07	0	94.82	78.57	0.314	26916	0.105

The last three columns indicate the average CPU times taken by SUBSET-LSLF, GA and LSMF respectively. Each row gives the average of CPU times taken by 10 problems in that problem set. We see that CPU times taken by SUBSET-LSLF and LSMF are much lower than that of GA. We can say here that SUBSET-LSLF and LSMF are computationally efficient because they are specially designed heuristics, which carefully and cleverly exploit the structure of these problems.

To improve the results further, we develop hybrid GA-LSLF and hybrid GA-LSMF by combining GA with SUBSET-LSLF and LSMF respectively. The results are shown in Table 2. Columns in Table 2 are similar to the columns in Table 1.

We find that the average percentage gap for GA-LSLF is very small (in the range of 0% to 0.78%). The average percentage gap is 0% for all the test problems in the case of GA-LSMF and thus the improvement over SUBSET-LSLF and GA-LSLF is 100%. This indicates that the GA-LSMF provides optimal solutions for all 150 test problems. It has to be noted that the algorithm takes advantage of both GA search process and problem specific information by LSMF to provide the optimal solution.

Note that the average CPU times taken by GA-LSMF are much lower than the other GAs, especially for large size test problems. In this case the GA starts with a very good solution obtained by LSMF and converges very fast.

FUTURE RESEARCH AND PROPOSAL FOR CONFERENCE PRESENTATION

We are currently working on testing these algorithms on a more diverse set of test problems for which the optimal values are not known. Research is also being conducted to improve the LSMF algorithm and its hybridization with GA. We have collected some real-world data and are analyzing the performance of these algorithms on this data. Using this data, we will illustrate how even a small improvement in the schedule can significantly impact the revenue generated from the ads.

Table 2. Comparison of Test Problems Results with Hybrid Gas

Prob Set #	No of slots (*N*)	Size of each slot (*S*)	%SUBSET-LSLF Gap			% Gap for Hybrid GA-LSLF (1)			% Gap for Hybrid GA-LSMF (2)			%Imp in Avg %Gap of (2) over SUBSET-LSLF	%Imp in Avg %Gap of (2) over (1)	Avg CPU Time for SUBSET-LSLF (in sec)	Avg CPU Time for (1) (in sec)	Avg CPU Time for (2) (in sec)
			Max	Avg	Min	Max	Avg	Min	Max	Avg	Min					
1	10	10	9	6.6	6	0	0	0	0	0	0	100	-	0.01	1.56	0.203
2	10	15	6.7	6.7	6.7	0	0	0	0	0	0	100	-	0.01	1.58	0.232
3	10	20	8	6.3	6	0	0	0	0	0	0	100	-	0.01	1.65	0.259
4	25	10	4.8	2.9	2.4	0	0	0	0	0	0	100	-	0.01	39.92	0.804
5	25	15	2.7	2.7	2.7	0	0	0	0	0	0	100	-	0.015	62.98	0.710
6	25	25	5.1	2.9	2.6	1.8	0.18	0	0	0	0	100	100	0.015	57.29	0.940
7	50	15	1.3	1.3	1.3	0	0	0	0	0	0	100	-	0.054	3984	4.009
8	50	25	3.2	2.1	1.3	0.9	0.33	0	0	0	0	100	100	0.059	4176	4.875
9	50	50	3.2	3.7	2.6	0.9	0.7	0	0	0	0	100	100	0.063	3966	9.485
10	75	25	2.6	1.5	0.9	0.9	0.63	0	0	0	0	100	100	0.154	11358	9.956
11	75	50	2.6	1.2	0.9	0.9	0.64	0.4	0	0	0	100	100	0.166	11718	14.41
12	75	75	0.9	0.9	0.9	0.9	0.78	0	0	0	0	100	100	0.167	11994	12.27
13	100	25	2.2	1.31	0.6	0.8	0.5	0	0	0	0	100	100	0.301	22332	17.87
14	100	50	2.2	1.6	0.6	0.7	0.59	0.3	0	0	0	100	100	0.347	23058	35.77
15	100	100	2.3	1.31	0.6	0.7	0.59	0	0	0	0	100	100	0.314	23598	22.78

References

Adler, M., Gibbons, P. B., and Matias, Y. "Scheduling Space-Sharing for Internet Advertising," *Journal of Scheduling*, 2001 (forthcoming).

Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S. "An Application of Bin-Packing to Multiprocessor Scheduling," *SIAM Journal of Computing,* (7:1), February 1978, pp. 1-17.

Dawande, M., Kumar, S., and Sriskandarajah, C. "Performance Bounds of Algorithms for Scheduling Advertisements on a Web Page," Working Paper. The University of Texas at Dallas, 2001.

Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

Kumar, S., Jacob, V. S., and Sriskandarajah, C. "Scheduling Advertisements on a Web Page to Maximize Space Utilization," Working Paper, The University of Texas at Dallas, 2001.

McCandless, M. "Web Advertising," *IEEE Intelligent Systems*, May/June, 1998, pp. 8-9.

Montgomery, D. C. *Design and Analysis of Experiments*, 4th edition, John Wiley & Sons, New York, 1996.

Novak, T. P., and Hoffman, D. L. "New Metrics for New Media: Toward the Development of Web Measurement Standards," *World Wide Web Journal (W3J*) (3:1), Winter 1997.

Rewick, J. "Choices, Choices: A Look at the Pros and Cons of Various Types of Web Advertising," *The Wall Street Journal*, April 23, 2001, p. R12.