*Int. J. Man–Machine Studies* (1984) **21**, 135–148

# On frequency-based menu-splitting algorithms

IAN H. WITTEN, JOHN G. CLEARY AND SAUL GREENBERG

*Man–Machine Systems Laboratory, Department of Computer Science,*
*The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4*

(*Received 1 February 1983, and in revised form 13 October 1983*)

If a menu-driven display is to be device-independent, the storage of information must be separated from its presentation by creating menus dynamically. As a first step, this article evaluates menu-construction algorithms for ordered directories whose access profile is specified. The algorithms are evaluated by the average number of selections required to retrieve items. While it is by no means suggested that the system designer should ignore other relevant information (natural groupings of menu items, context in terms of prior selections, and so on), the average selection count provides an unambiguous quantitative criterion by which to evaluate the performance of menu-construction algorithms.

Even in this tightly-circumscribed situation, optimal menu construction is surprisingly difficult. If the directory entries are accessed uniformly, theoretical analysis leads to a selection algorithm different from the obvious one of splitting ranges into approximately equal parts at each stage. Analysis is intractable for other distributions, although the performance of menu-splitting algorithms can be bounded. The optimal menu tree can be found by searching, but this is computationally infeasible for any but the smallest problems.

Several practical algorithms, which differ in their treatment of rounding in the menu-splitting process and lead in general to quite different menu trees, have been investigated by computer simulation with a Zipf distribution access profile. Surprisingly, their performance is remarkably similar. However, our limited experience with optimal menu trees suggests that these algorithms leave some room for improvement.

## Introduction

Menu-based selection is becoming a popular man–machine interface technique for the casual user. Public information systems like Prestel, Antiope, and Telidon use it to enable a customer to browse through a large information structure. Although direct access to a page by its page number is usually permitted, and some more sophisticated methods such as keyword searches (Bochmann, Gecsei & Lin, 1982) have been explored, menu-selection remains the primary interface technique in these systems.

A deficiency in current implementations of casual-user information systems is that the menus and page numbers are built into the database. Pages are formatted before being written into the information structure. For example, when a large list like a telephone directory is to be made available, the search tree must be pre-calculated and formatted into fixed pages.†

This effectively locks the entire database into the display technology that is used. It is then impossible to tailor the system to different display devices. For example, one cannot profit from any extra lines on a larger screen by lengthening the menus. Neither

† Kwok (1982) gives a description of such a scheme for a Chinese videotext system.



0020-7373/84/080135+14$03.00/0 © 1984 Academic Press Inc. (London) Limited

can one cope in a natural manner with a smaller screen (say, one with large characters for the visually handicapped). It is very difficult to map the selections on to a different medium such a voice output, where the "display" is ephemeral and good menus therefore much smaller.

There is clearly a case for storing a higher-level representation of information in such a system, and performing the formatting operation on demand when a page is viewed. This allows account to be taken of different display device characteristics. It also encourages the use of frequency-based techniques to hasten access to often-used entries at the expense of rarer ones.

This article discusses the problem of menu construction in the context of a directory look-up system. The average number of selections required to retrieve items is used to evaluate menu-construction algorithms. While it is by no means suggested that the system designer should ignore other relevant information (natural groupings of menu items, context in terms of prior selections, and so on), the average selection count does provide an unambiguous quantitative evaluation criterion. In fact, even in such a constrained situation the process of menu construction is surprisingly nontrivial, and optimal algorithms require a prohibitively large amount of search.

The first section sets the scene by describing the menu-based look-up method. Menu selection for a directory whose members are accessed equally often is examined next. This special case is amenable to theoretical treatment. The Zipf distribution is used subsequently as a plausible model of many menu-based activities. After a brief summary of germane properties of this distribution, several algorithms for menu splitting are considered, and results of experiments described which evaluate them on the basis of the average number of selections, or keystrokes, required to access an item. Finally, some conclusions are drawn for practical implementations.

## Retrieval from a directory

By way of example, suppose that it is desired to retrieve a telephone number from a directory on a menu-based information system. The directory has $N$ entries, and the menu can display $M$ items. The presentation system displays a *range* of names as each menu item. This divides the name space into $M$ regions, and the user is invited to choose one as his first-level selection. The system will take the indicated range, split it into $M$ parts, and await a further selection. Eventually, some or all of the ranges will have converged on to unique entries, and the final selection will indicate which one is sought. At this stage the system may auto-dial the associated telephone number. However, the present work studies the selection process itself, and what happens at the leaf is not of immediate concern.

Figure 1 shows a menu tree for a directory of 20 names, produced by subdividing the name space as equally as possible at each stage with a menu size of 4. An example of the corresponding first-level menu is shown in Table 1.

A real telephone directory system might have $N = 250{,}000$ and $M = 16$. Using uniform subdivision, the retrieval process would take four or five selections, a cost which the user may deem reasonable for infrequent entries but too high for frequently-called numbers.

In most cases the act of selection provides information about numbers that are likely to be selected in the future. In a limited field study of telephone usage, one of the

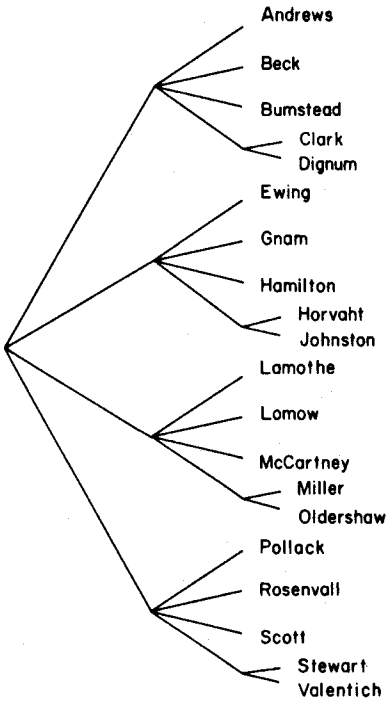


FIG. 1. Menu tree generated by uniform subdivision with $N = 20$, $M = 4$.

TABLE 1
*First-level menu for Fig. 1*

| | |
|---|---|
| 1 | Andrews–Dignum |
| 2 | Ewing–Johnston |
| 3 | Lamothe–Oldershaw |
| 4 | Pollack–Valentich |

authors (Greenberg) discovered that almost 50% of his subjects' calls were to numbers they had called previously. The likelihood of a selected number being repeated in future is quite high, especially when compared with the majority of nonselected entries. As far as retrieval is concerned, this information can be modelled as a non-uniform distribution over the name space, in which popular entries are given high probabilities. Crucial parameters determining the actual distribution are the maximum number of selections the user is willing to undergo to access a new item, the time-constant of decay for once-popular but now disused entries, and the amount of reinforcement given to entries when they are selected.

A menu tree may alter dramatically over time to reflect the different popularity of entries. For instance, Fig. 2 represents a tree in which Andrews and Beck are called often, Bumstead through Ewing are called infrequently, and the rest are almost never called. Table 2 shows the top-level menu representation of this tree. Menu items 1 and 2 indicate unique names. If they were selected, any action (such as auto-dialling)

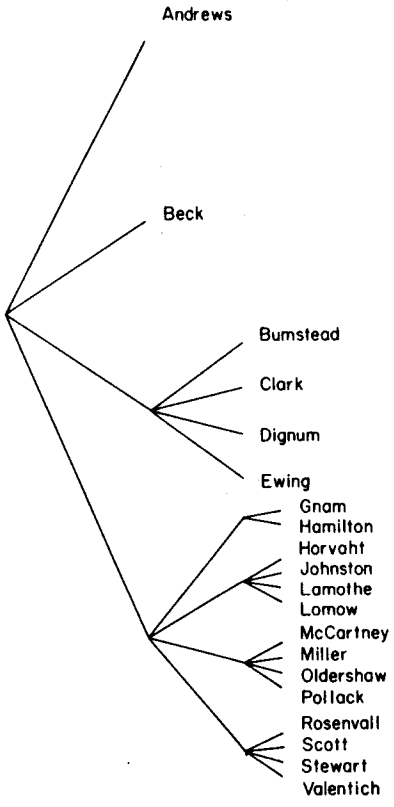


FIG. 2. Menu tree for a monotonically decreasing distribution ($N = 20$, $M = 4$).

TABLE 2
*First-level menu for Fig. 2*

| | |
|---|---|
| 1 | Andrews |
| 2 | Beck |
| 3 | Bumstead–Ewing |
| 4 | Gnam–Valentich |

could take place immediately. Menu item 3 leads to only four entries, which will be disambiguated by the second-level menu. Menu item 4 covers the majority of the name space, for which three selections are required.

Adjusting the menu tree in this way to reflect actual usage can reduce the average number of selections needed to retrieve entries. It presents two problems: splitting the distribution and updating the usage probabilities. The present paper addresses the first of these; some discussion of the second can be found in Witten, Greenberg & Cleary (1983).

## The uniform distribution

The obvious way to construct the menus is to split the directory into $M$ equal parts and invite the user to select one of them. Then, that part in turn is split into $M$ equal parts, and so on. This procedure, which is illustrated by Fig. 1, is called "algorithm A". It will take between $k$ and $k+1$ selections, where $k$ is the largest integer with

$$M^k \leq N;$$

that is,

$$k = \left\lfloor \frac{\log N}{\log M} \right\rfloor.$$

If each item in the directory is accessed equally often, the mean number of selections to retrieve an item will be

$$s = k + \text{Min}\ \{1,\ 2(1 - M^k/N)\}.$$

When $N$ is between $2M^k$ and $M^{k+1}$, all the level-$k$ menus will have two or more entries and so all accesses will require $s = k+1$ selections. Between $M^k$ and $2M^k$ some accesses will require $k$ selections and others $k+1$. As the level-$k$ menus fill up the average number of selections increases and so $s$ is a monotonically increasing function of $N$.

For a uniform distribution, the average selection count can easily be reduced below that achieved by algorithm A. To do so, it is necessary to abandon the obvious simplification of considering each level of menu splitting in isolation from the others. (While this is easy in the case of the uniform distribution, it leads to severe computational difficulties when we consider the general case below.)

The improvement is illustrated by Figs 1 and 3, which show a simple example with $N = 20$, $M = 4$. A two-level tree discriminates 16 nodes, and algorithm A, shown in Fig. 1, will split four of these into pairs and use the remaining 12 as leaves. Thus eight items require three selections, while the remainder are found in two; giving an average of $2 \cdot 4 = 48/20$ selections per item.

However, it is obvious that many potential leaf nodes are unused, because the menu size is $M = 4$ but the lowest leaves derive from a two-way split only. It is more economical to split the smallest possible number of level-2 nodes four ways, as shown in Fig. 3. Then only two of the level-2 nodes need be split, so that six items require three selections and the remaining 14 require two. This gives an average of $2 \cdot 3 = 46/20$ selections per item.

This is clearly the best possible menu-selection method for a uniform distribution. It is easy to calculate the average number of selections it requires. With $k$ as defined above, the depth of the tree is between $k$ and $k+1$. Let $q$ be the number of leaves at level $k$. Then the remaining $M^k - q$ level-$k$ nodes can be expanded into $(M^k - q)M$ nodes at level $k+1$. It is best to choose $q$ as the largest integer with

$$q + (M^k - q)M \geq N;$$

in other words,

$$q = \left\lfloor \frac{M^{k+1} - N}{M - 1} \right\rfloor.$$

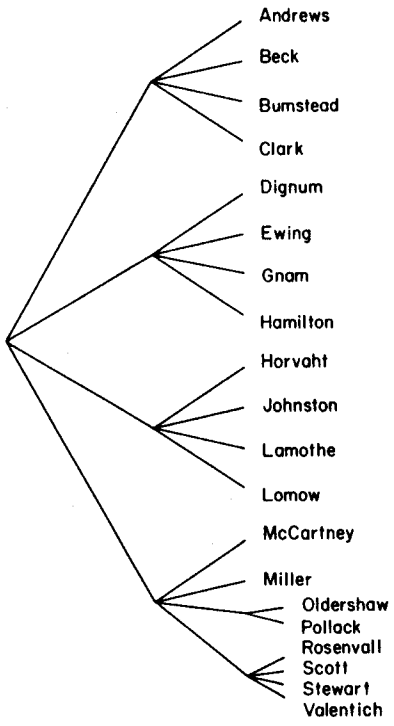


FIG. 3. Optimal menu tree for a uniform distribution ($N=20$, $M=4$).

From this it can be shown that the average number of selections to retrieve an item is

$$k+1-\frac{1}{N}\left\lfloor\frac{M^{k+1}-N}{M-1}\right\rfloor.$$

Figure 4 shows the performance of algorithm A (upper line) and this optimal algorithm (middle line), with $M=4$. The vertical axis gives the average number of selections required, while the horizontal one shows $N$ on a logarithmic scale.

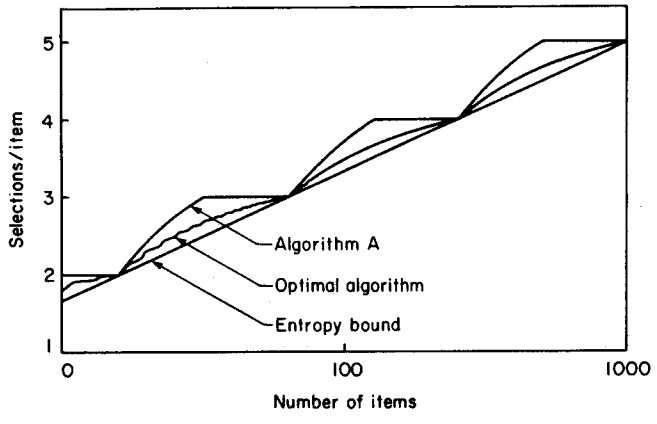


FIG. 4. Average selection counts for a uniform distribution (4-item menus).

## Non-uniform distributions

Items in a directory are not necessarily accessed equally often. And if the access frequencies are known, it is possible to design a menu-splitting algorithm which takes these frequencies into account and so reduces the average number of menus which are displayed when accessing items. Suppose the access frequency of word $i$ is $p_i$, where

$$\sum_{i=1}^{N} p_i = 1.$$

Then the entropy of an item is defined as

$$-\sum_{i=1}^{N} p_i \log p_i \text{ bits.}$$

(All logarithms indicated by "log" are to base 2; natural logarithms are written "ln".)

Now consider the process of retrieving items from the directory by menu-selection with a menu of size $M$. If $s_i$ is the number of selections required to access word $i$, it follows that

$$\sum_{i=1}^{N} s_i p_i \log M \geq -\sum_{i=1}^{N} p_i \log p_i,$$

or

$$s \geq -\sum_{i=1}^{N} p_i \log p_i / \log M,$$

where $s$ is the mean number of selections needed to access an item.

This lower bound for $s$ is smaller—often much smaller—than the expression given above for algorithm A. It is shown as the bottom line in Fig. 4 for a uniform distribution. However, as the figure shows, it is not usually achievable because of inevitable quantization effects in the menu-selection process. We cannot exhibit a tighter lower bound in closed form, in general. The remainder of this article examines how closely the above bound can be approached by practical menu-selection algorithms.

## The Zipf distribution

The Zipf probability distribution for directory accesses has been chosen in order to evaluate selection algorithms. This approximates word usage statistics in natural language (Zipf, 1949), and forms a good model of some more artificial phenomena (Peachey, Bunt & Colbourn, 1982). It can be defined in terms of the probability assignment

$$p_{\pi(i)} = \mu / i,$$

under a permutation $\pi$ of the integers $1, 2, \ldots, N$. Probability normalization gives the constant $\mu$ as

$$\mu \approx \frac{1}{\ln N + \gamma} \quad \text{for large } N,$$

where $\gamma$ is Euler's constant, 0·57721....

As a modest example of directory usage patterns, a month-long study was made of personal telephone usage. The numbers called by several subjects were recorded and ranked by frequency, and the results for the three most active are plotted in Fig. 5. The vertical axis shows the number of calls, normalized to one for the most popular telephone number; while the horizontal axis shows the rank ordering of popularity.

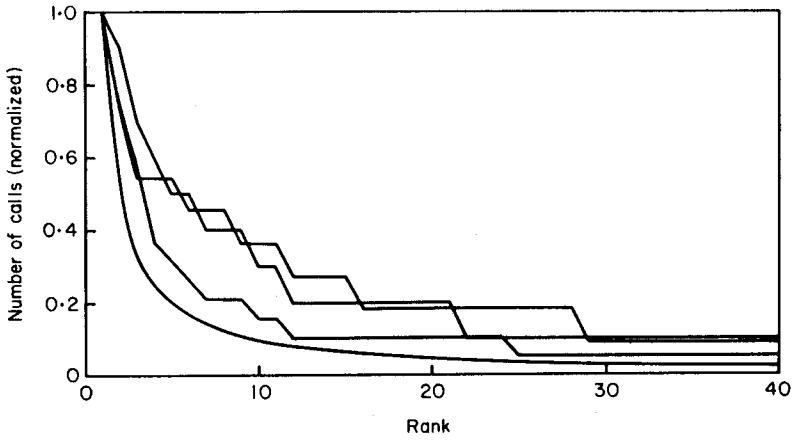


FIG. 5. Sample distributions of personal telephone usage, compared with Zipf.

The continuous curve is a true Zipf distribution, normalized in the same way. The same decreasing trend can be seen in each usage pattern, although it is significantly less pronounced than in the Zipf distribution. Nevertheless, the Zipf distribution provides a plausible model of the repetition effects found in highly-variable real-life usage patterns.

A crude approximation to the entropy of this distribution, for large $N$, is

$$-\sum_{i=1}^{N} p_i \log p_i \approx \frac{\mu}{\ln 2}[\tfrac{1}{2}\ln 2+\tfrac{1}{3}\ln 3+\tfrac{1}{2}[\ln (N+\tfrac{1}{2})]^2-\tfrac{1}{2}[\ln \tfrac{7}{2}]^2]-\log \mu.$$

This is asymptotic to

$$\tfrac{1}{2}s+\log s-0{\cdot}9451+O(1/s),$$

where $s=\log N$. This indicates that there may be potential for reducing the average number of selections to half that required for the simple frequency-independent menu-splitting technique, if the directory is sufficiently large.

## Menu-splitting algorithms

The performance of three menu-splitting algorithms has been tested on directories of various sizes, all under the Zipf distribution. They operate by attempting to make a good split at each level of the selection process, in isolation from the others. However, it is not obvious how best to tackle the inevitable discreteness problem, where a split cannot be made at exactly the point in the probability distribution where one would like it to be because a single high-probability item broadly straddles that point. The

three algorithms take different approaches to this rounding problem, and we conjectured initially that they would exhibit radically different behaviours.

This conjecture has turned out to be false. The three methods give very similar results. To make matters worse, as shown below, the average number of selections is significantly above the entropy-based lower bound discussed earlier, using any of the methods. Almost all the difference is due to the policy of considering each level in the selection process in isolation from the others. As seen above for the uniform distribution, this can give significantly poorer performance than the optimal splitting strategy because terminal menus are not fully utilized.

To investigate the matter further, a procedure, algorithm O, has been constructed which guarantees to find the best sequence of menus to minimize the average selection count. The program searches through possible menu trees. For example, it produces the tree of Fig. 3 (and discards that of Fig. 1) in the uniform distribution case. Unfortunately, for non-uniform distributions the search space is so large that the procedure consumes a great deal of space and time, and is only feasible for unrealistically small examples.

Because algorithm O can only be tested on small problems, the question is still open as to how much room is left for improvement between the three simple algorithms and the theoretical entropy-based lower bound. This makes it interesting to consider whether the bound can be tightened. It is not hard to see that given a set of probabilities to be attached to directory entries, the most favourable condition for a menu-selection algorithm is when the probabilities are monotonically decreasing (or increasing) throughout the directory. If this is the case, the optimal tree will have the property that its depth increases, weakly, from top to bottom (or vice versa). Figure 2 shows such a tree with leaves at depths 1, 2, and 3; while Fig. 3 provides another example where the leaf depths are 2 and 3. For a tighter lower bound on the mean selection count attainable with a Zipf distribution, therefore, it is sufficient to consider monotonically decreasing distributions and trees of the form shown in Figs 2 and 3. For such distributions and trees, the search for the optimal tree is greatly reduced, so that it now becomes feasible. For all examples below, this bound, referred to as the *monotonic bound*, is shown as well as the looser entropy-based one. It seems from the examples that the entropy bound is quite tight.

### ALGORITHM B: ITERATIVE SPLITTING

This method splits the directory into $M$ parts which have approximately equal total probabilities. It does so by advancing through the directory, and making the first split at as close as possible to $1/M$ of the total probability. The second split is made as close as possible to $1/(M-1)$ of the way through the probability remaining after the first split, and so on. This iterative technique was chosen to minimize the effect of inexact early splits on subsequent split points. Once the directory has been divided into $M$ parts, the appropriate one is selected and the range split again. The process continues until the item sought appears in a part by itself.

### ALGORITHM C: RECURSIVE SPLITTING

Another way of minimizing the effect of inexact splits on later decisions is to split the range recursively. At first, the $M/2$ point is sought. (If $M$ is odd, either the $(M-1)/2$ or the $(M+1)/2$ point is selected instead, at random; this requires a small but obvious

modification to the procedure described.) This is done by splitting the probability range into two parts which are as nearly equal as possible. Then the two parts are further split, recursively, until all $M$ splits have been made. As before, once the directory has been divided into $M$ parts, the appropriate one is selected and the range split again; the process terminating when the sought item appears by itself.

ALGORITHM D: OPTIMAL LEVEL-BY-LEVEL SPLITTING

This algorithm promises to find the optimal way of splitting the menu at each stage of the menu-search procedure to maximize the entropy of each selection, taken in isolation from the others. The problem is one of dividing an ordered list containing $N$ numbers among $M$ buckets so as to maximize the entropy of the resulting lumped distribution.

TABLE 3
*Candidate split points for a 2 : 3 division of an 8-item menu*

| Item | Probability | Candidate split points | Entropy (upper bound) |
|---|---|---|---|
| 1 | 3/15 | | |
| 2 | 2/15 | | |
| | | ——— | 2·31 |
| 3 | 1/15 | | |
| | | ——— | 2·32 |
| 4 | 2/15 | | |
| | | ——— | 2·27 |
| 5 | 2/15 | | |
| | | ——— | 2·11 |
| 6 | 1/15 | | |
| 7 | 1/15 | | |
| 8 | 3/15 | | |

The method is best explained in the context of an example. Table 3 gives the probabilities of $N = 8$ items, to be displayed on a menu of size $M = 5$. Consider initally how to divide the menu into two regions, the first of which will eventually be split again into two and the second into three, to make a total of $M = 5$ regions. The division should clearly not be made between the first two items, for then it will not be possible subsequently to divide the first region. Similarly, it should not be between the sixth and seventh, nor between the seventh and eighth; for the second region must eventually be split into three parts. Table 3 shows the candidate division points. For each of these, an upper bound to the entropy of the final lumped distribution can be calculated as follows.

Consider the first candidate shown in the table. It divides the total probability into 5/15 and 10/15. The first region will be split again, dividing the probabilities into at best 5/30, 5/30. In fact it is obvious that the ultimate division will not be as good as this, for in this case it can only split the first two items and result in selection probabilities of 3/15, 2/15. But an upper bound is sought. The second region will be split again into three parts, giving (at best) probabilities of 10/45, 10/45, 10/45. Thus an upper bound to the entropy ultimately realized by taking the first candidate can be obtained from the set of probabilities {5/30, 5/30, 10/45, 10/45, 10/45}—a numerical value of 2·31. The same procedure gives the other numbers in the table.

The most promising choice for the 2:3 split is between items 3 and 4. Now the process has to be repeated for each sub-part. Unfortunately, quantization effects may mean that there is actually a better place than this for the 2:3 split. It is necessary to perform a recursive search operation for the true optimal split points. This could involve, for example, taking the points adjacent to the most promising choice and investigating them as well. The guaranteed upper bound, which decreases monotonically on either side of its maximum, provides an excellent basis for pruning the search tree.

Unfortunately, even when complete, this algorithm only optimizes the individual menus. It does not optimize the *sequence* of menus which was seen above to be so important for the uniform distribution. For the Zipf distribution also, it appears that the best sequence of menus is invariably considerably different from the best menus at each stage.

## Performance of the algorithms

Figure 6 shows the performance of the algorithms on directories of up to $N = 1000$ entries, with $M = 4$. Graphs are given for the entropy-based lower bound,

$$-\sum_{i=1}^{N} p_i \log p_i / \log M;$$

the tighter monotonic bound; algorithms B, C, and D; and algorithm A which ignores the probabilities altogether. Each point on each of the graphs for algorithms A–D represents the result of simulation on twenty randomly-chosen Zipf distributions differing only in the order of the probabilities. The kinks which appear on the increasing parts of the algorithm A curve are due to the small size of this sample.

As can be seen, the performance of algorithms B, C, and D is virtually identical. All perform significantly better than algorithm A, and significantly worse than both lower bounds—which are surprisingly close together. None of the three performs

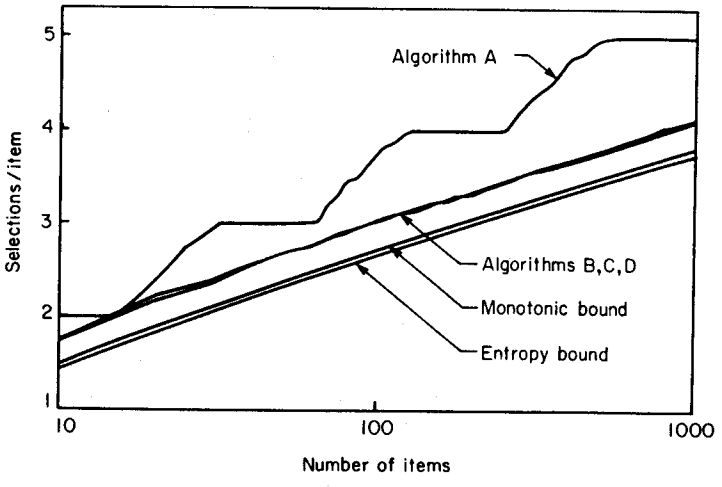


FIG. 6. Mean selections vs directory size for 4-item menus.

uniformly better than the others. The fully-optimal algorithm O is not represented because it is computationally infeasible to run on directories of this size.

It would be fascinating to know to what extent the gap between the monotonic bound and the algorithms could be reduced by clever procedures. Figure 7 shows the effect of varying the menu size for a small directory of $N = 20$ items—small enough for algorithm O to be feasible. The menu size varies from 2 to the maximum of 20. All methods naturally require only one selection when the menu is as large as the directory. However, the entropy-based lower bound is less than unity at this point, reflecting the fact that the entropy of a selection from a Zipf distribution is less than one from a uniform distribution.

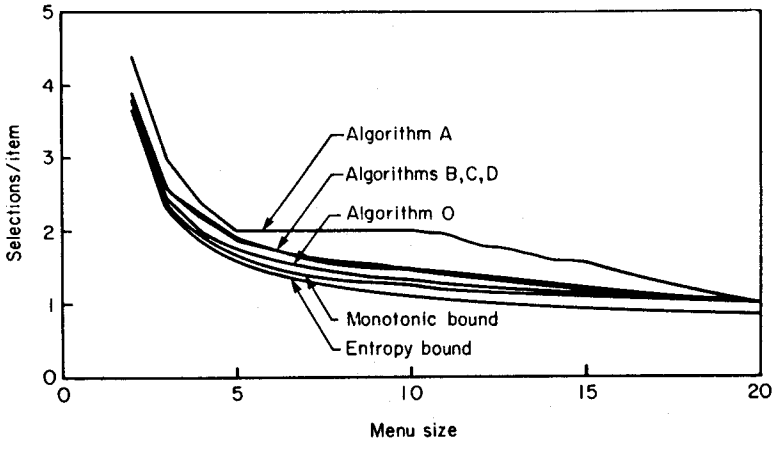


FIG. 7. Mean selections vs menu size for a 20-item directory.

From the bottom, the graphs of Fig. 7 represent the entropy-based lower bound; the monotonic bound; algorithm O; algorithms B, C, and D (together); and algorithm A. Algorithm O produces results around midway between the tightest, monotonic, bound and algorithms B, C, and D. It is concluded that there is some room for improvement over these methods.

One may wonder how the algorithms behave for menus larger than the $M = 4$ of Fig. 6, on directories of realistic size. Figure 8 shows results with $M = 10$ and $N$ ranging up to 10,000. Here, algorithm D is not represented because even it requires excessive computational resources for menus of 10 items or more, involving as it does recursive searches over alternative candidate divisions.

These graphs are very similar to those of Fig. 6. The gap between algorithms B and C, and the lower bounds, grows as $N$ increases, but only very slowly. As before, algorithms B and C represent worthwhile improvements over A, and further improvements may be possible. Algorithm B—iterative splitting—is, in fact, consistently superior to C—recursive splitting—for directories of $N = 750$ and above, although only by a small margin. This is also true for the results shown in Fig. 6 with $M = 4$, for $N = 250$ or more.

All these results represent the average of 20 randomly-chosen Zipf distributions differing only in the order of probabilities. As well as this mean value, the spread

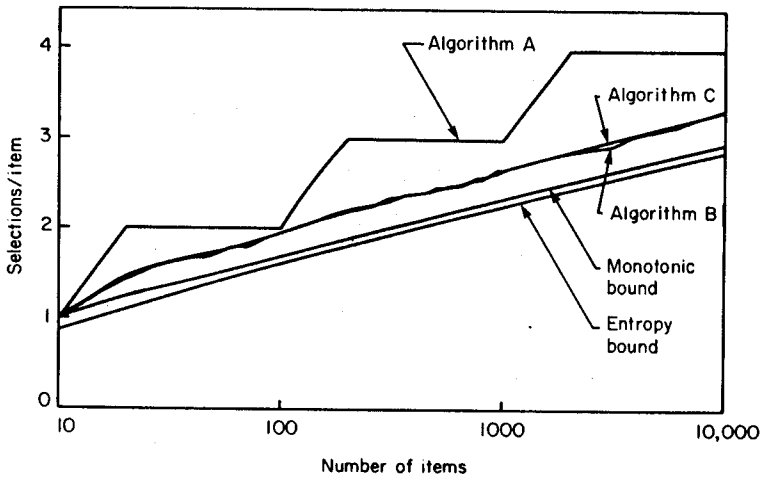


FIG. 8. Mean selections vs directory size for 10-item menus.

between the maximum and minimum selection counts is of interest. If results were occasionally found which were much worse than average, the robustness of the menu-searching procedure would be in question, for the user could find himself with an "unlucky" probability distribution. However, our experiments indicate that this is not likely to be so, for any of the algorithms. In the situation of Fig. 6, the average number of selections to retrieve an item varied by around 3%, depending on which of the 20 Zipf distributions was chosen; while for Fig. 8 the difference was around 7%. These differences were about the same for each of algorithms B, C, and D. In no case was a difference of more than 10% observed.

## Conclusions

Several algorithms for selecting items from an ordered directory with specified probabilities have been investigated. They all involve the selection of ranges of directory entries from a succession of menus, the range narrowing progressively until it contains just the desired item. The aim is to minimize the average number of menu selections which are needed to find an item.

In the case of uniformly-distributed directory entries, which is amenable to theoretical treatment, the simplistic method of splitting each range into as many parts as there are menu items was found to be inferior—often grossly inferior—to a method which instead maximizes the utilization of leaves in the menu tree.

For the Zipf distribution, theoretical analysis is intractable. A lower bound can be derived from entropy considerations. The optimal menu tree can be discovered by searching, but this is computationally infeasible for any but the smallest problems. A somewhat tighter bound can be found by searching for the optimal menu when the directory entries are arranged in the best possible way, that is, ordered according to their probability. Then the search space is much reduced and the optimal menu tree can be easily obtained.

Three algorithms, B, C, and D, which split the range into menu items of approximately equal probability, were investigated. In general, they lead to quite different menu trees

because of the large effect of rounding in the menu-splitting process. One of them, D, maximizes the entropy of each selection individually by dividing the range in the best possible way: although this in no way guarantees the optimality of the *sequence* of selections. Despite this, however, the performance of all three was found by simulation to be remarkably similar.

The small experience we have with the optimal menu tree suggests that there is some room for improvement over these algorithms. We predict that this will be obtained from heuristic methods which concentrate more on fully-utilizing leaf nodes rather than on dividing the probability range as equally as possible. Until better methods are found, however, the simple process of iteratively splitting the probability range at each stage into regions whose probabilities are similar as possible appears to be as good as any. Indeed, it produces marginally better results for large directories than the recursive splitting technique.

We would like to thank an anonymous referee for suggestions which helped to improve the presentation of this article. The research is supported by the Natural Sciences and Engineering Research Council of Canada.

## References

BOCHMANN, G. V., GECSEI, J. & LIN, E. (1982). Keyword access in Telidon: an experiment. *Proceedings of Videotex 82,* New York (June).

KWOK, P. C. K. (1982). Man–machine interaction in telephone directory enquiry in Videotex systems. *Proceedings of IEE Conference on Man–Machine Interaction,* Manchester, England, pp. 51–54 (July).

PEACHEY, J. B., BUNT, R. B. & COLBOURN, C. J. (1982). Bradford–Zipf phenomena in computer systems. *Proceedings of Canadian Information Processing Society National Conference,* Saskatoon, Saskatchewan, pp. 155–161 (May).

WITTEN, I. H., GREENBERG, S. & CLEARY, J. (1983). Personalizable directories: a case study in automatic user modelling. *Proceedings of Graphics Interface 83,* Edmonton, Alberta, pp. 183–189 (May).

ZIPF, G. K. (1949). *Human Behaviour and the Principle of Least Effort.* Ontario: Addison–Wesley.