Recall that our query formulation already contains subrequests consisting of descriptors D, I, and J. Hence, at this point the query formulation will be as follows:

$$D \vee I \vee J \vee (G \wedge L \wedge C) \vee (B \wedge L \wedge C) \vee (M \wedge B \wedge C) \vee (B \wedge G \wedge C) \vee (B \wedge G \wedge L) \vee (M \wedge G \wedge C)$$

After analyzing all possible combinations of three descriptors and discarding extraneous subrequests, if they exist, the algorithm looks at the descriptors with $\Psi_i$ values in the next interval on the left and constructs subrequests consisting of four descriptors going through the steps described previously. This process will continue for all existing intervals. All constructed subrequests will be added to the query formulation and after the last interval is analyzed the final version of the query formulation will be used for the search.

Notice that the algorithm described earlier allows the system to automatically translate a search request into the information retrieval language, even in a case where the user formulates a search request without using a marked set of documents, that is, in a traditional natural language form. Then the search request is considered by the algorithm as a marked set consisting of one document and the algorithm proceeds as noted earlier. In this case the matrix becomes a one-dimensional array and the choice of bound values is performed automatically as described next. It should be mentioned that the quality of the search based on the query formulation obtained from one document is typically inferior to the search based on the query formulation obtained from several documents. The steps for constructing a query formulation are given in Table 7.1.

**Table 7.1**
Steps in Constructing a Query Formulation

1. Construct the relevant neighborhood (by indexing the incoming search request).
2. Construct the term-document matrix.
3. Compute $\Psi_i$ values for all descriptors in the relevant neighborhood.
4. Compute the bound values for the intervals where each interval determines how many descriptors will be combined in one subrequest.
5. Find all the descriptors whose $\Psi_i$ values are greater than L. Each of these descriptors will constitute a separate subrequest in a final query formulation.
6. Remove all descriptors found in step 5 from the relevant neighborhood, transform the term-document matrix, and place all remaining descriptors in corresponding intervals.
7. Analyze every interval and construct subrequests corresponding to each interval (using the Boolean operator AND).
8. Remove extraneous subrequests.
9. Combine all remaining subrequests into final query formulation using the Boolean operator OR.



## 7.5 The Choice of Bound Values

From the description of the preceding algorithm, it is clear that the choice of the bound values

$$(L, L_1, L_2, \ldots, L_n)$$

determines the quality of the query formulation (given two query formulations Q1 and Q2, we say that Q1 is "better" than Q2 if the search based on Q1 gives better results than the search based on Q2, for example, using recall and precision levels for comparison). If we increase the bound values, the number of subrequests in our query formulation decreases, but at the same time we may increase the probability that the documents found during the search will correspond to the information need of the user. In other words, by making the bounds more strict, we increase the precision of the search but decrease the recall level. It is clear that the ranges of the bound values for the intervals, the maximum number of descriptors allowed in a subrequest, and the required occurrence frequency of the set of descriptors in the documents of the marked set either have to be defined in advance or a method has to be established that would allow the algorithm to compute these values automatically.

When the values are entered into the system in advance, the developers of the system could determine these values (as we did it earlier) on the basis of the required precision and recall levels, using the characteristics of the collection of documents (for example, the average length of the document profile), the requirements of individual users, and other factors. In Figure 7.6 the bound values were determined for a collection of documents consisting of 2504 abstracts from computer science with the average length of an abstract profile approximately equal to 12 descriptors. Obviously there is no reason to believe that the chosen bound values were optimal. But these values were used as the basis for subsequent experiments, and as a result of these experiments we found a method whereby the bound values are determined automatically by a new algorithm.

The algorithm computes the bound values for each individual user. We assume that a user provides the system with the information about the number of documents that should be contained in the output. For example, when the user begins a retrospective search from a collection containing 30,000 documents, he or she might indicate that the output should contain 50 documents; from a collection of 1000 documents, the user might request an output of 10 documents; and so on. In both cases the algorithm uses some initial bound values (see, for example, Figure 7.6) to obtain the query formulation that is going to be used for the initial search. But the results of the search are not given to the user and are used by the algorithm for further calculations. The algorithm compares the number of documents found during the initial search with the number of documents requested by the user. If the two numbers are equal, then