the file. This uniqueness is an explicit design decision, not because it is technically impossible to do otherwise. The records in a relative file are numbered from 1 to n, where n is defined as the highest possible record number in the file. The user data records must fit within a cell and cannot overlap between multiple cells. Therefore, the cells themselves are numbered from 1 to n. When the user defines a relative file, one of the parameters that can be set is n, known as the maximum record number. This number is of course the highest possible record number that the relative file access method can accept as being valid.

The formula for locating the cell with the target record is given below. Let x denote number of cells per block and r denote the target record number:

$$\text{block number} = \text{integer}\left(\frac{r - 1 + x}{x}\right);$$

$$\text{cell number in block} = 1 + \text{mod}\left(\frac{r - 1 + x}{x}\right).$$

Thus, to locate any record in a relative file, the access method must first convert the record number into a block and cell number within the file. Then, at the most, one I/O operation will have to be performed in order to read the target block into primary storage.

Now having considered the random access principle (taking a relative file as an example), let us make two substantial additions. The first addition concerns the performance of the IR system utilizing relative file organization. It is not accidental that the address of a record with the text from the text file (rather than the record number) is stored in field 1 of a corresponding record from the document profile file. Even at the stage of organizing a file document profile, it makes sense to immediately store the result of the above calculations, rather than the ordinal number of a document, into field 1. This result is an actual address made up of the block number and the cell number in the text file. In this case, during a search process (repeated for many search requests), there is no need to calculate the address, which improves the system's performance.

Another addition concerns the very calculation of the address. We have already seen that calculation makes possible the so-called one-disk-access method. In computer science such a calculation is called *hashing* (see, for example, Folk & Zoellick, 1992; Horbron, 1988; and Wartik, Fox, Heath, & Chen, 1992). A *hashing algorithm* or *hashing function* is a means of calculating the disk address of a block, and in our case also of a cell, containing a given record from the value of its key. Sometimes hashing is called key-to-address transformation. A hash function is like a black box that produces an address every time you drop a key. More formally, it is a function $h(K)$ that transforms a key K into an address. There are many hashing algorithms available that distribute the records within the file. In this sense the preceding formulas only illustrate implementation of one of the possible hashing algorithms. Hatching is commonly

considered to be an extraordinary discovery in computer science, one providing methods that yield truly amazing performance advantages. Indeed, this method has gained wide acceptance, though it is known to have certain drawbacks.

After discussing the random access principles, we may proceed to a more detailed examination of the second part of the search program, that is, the part responsible for the address search. Recall that as the result of the subject search, a collection of retrieved addresses is formed in primary storage and that every address consists of the block number and the cell number. It is precisely these addresses that underlie the search for addresses. This search begins with grouping the addresses selected during the subject search in primary storage. In other words, action is taken to determine if there are retrieved addresses giving the same block number. This type of grouping can enhance performance because to retrieve texts from such groups it is sufficient to read only the respective (one) block for a group of (searched) records contained in this block. Then blocks are retrieved from the file. The first blocks retrieved are those corresponding to the groups available, and from each block only those records (texts) are read that were found during the subject search. In primary storage an output begins to form of the read texts. The next blocks retrieved are those corresponding to the ungrouped addresses. The respective texts are also read out of these blocks, and after reading the last text from the last block, the output is considered formed and ready for the user.

Now, if this two-stage pattern of search is compared with the single-stage pattern given to clarify the principle of sequential access, then it can be considered sufficiently workable. At least the speed of formation of the output has increased significantly. Still, the relative file has a number of drawbacks, and in IR systems the most important one is poor disk space utilization. Indeed, the size of every cell must be equal to the size of the longest record in the file. In our example all cells must be equal to the longest of the 10,000 available texts. Even if the input texts are assumed to be only abstracts (which is quite realistic because only abstracts are utilized in many real systems) and among the 10,000 there is one that is ten times as large as the average abstract in the file, then 90% of the memory occupied by the file will be empty space. Nevertheless, when the subject search is supposed to use sequential file organization, then the described approach is quite promising.

We said, "when the subject search is supposed to use sequential file organization." But is it possible to do without sequential access if all available document profiles are supposed to be compared with the query formulation? For if it is necessary to read and then compare all the records, there is simply nothing better than sequential file organization. It turns out, however, that in the Boolean system one can do without sequential access; that is, in organizing a subject search in the Boolean system one can avoid comparing every document profile and query formulation. Moreover, none of the document profiles containing descriptors that correspond to a subrequest from the available query formulation