16 (polycyclic training set), 224-41-9; **16** (benzodiazepine), 846-58-2; **17** (polycyclic training set), 188-52-3; **17** (benzodiazepine), 1622-61-3; **18** (polycyclic training set), 191-26-4; **18** (benzodiazepine), 4980-73-8; **19** (polycyclic training set), 191-24-2; **19** (benzodiazepine), 2011-67-8; **20** (polycyclic training set), 191-30-0; **20** (benzodiazepine), 2891-09-0; **21** (polycyclic training set), 189-55-9; **21** (benzodiazepine), 5527-71-9; **22** (polycyclic training set), 192-51-8; **22** (benzodiazepine), 30144-75-3; **23** (polycyclic training set), 196-42-9; **23** (benzodiazepine), 846-50-4; **24** (polycyclic training set), 193-09-9; **24** (benzodiazepine), 1812-30-2; **25** (polycyclic training set), 227-04-3; **25** (benzodiazepine), 2894-68-0; **26** (polycyclic training set), 216-00-2; **27** (polycyclic training set), 195-00-6; **28** (polycyclic training set), 222-78-6.

REFERENCES AND NOTES

(1) Klopman, G. *J. Am. Chem. Soc.* **1984**, *106*, 7315.
(2) Stuper, A. J.; Brugger, W. E.; Jurs, P. C. *Computer Assisted Studied of Chemical Structure and Biological Function*; Wiley-Interscience: New York, 1979.
(3) Kier, L. B.; Hall, L. H. *Molecular Connectivity in Structure-Activity Analysis*; Wiley-Research Studies Press, Letchworth, U.K., 1986.
(4) Bonchev, D. *Information Theoretic Indices for Characterization of Chemical Structures*, Wiley-Research Studies Press, Chichester, U.K., 1983.
(5) Chu, K. C.; Feldman, R. J.; Shapiro, M. B.; Hazard, G. F., Jr.; Geran, R. I. *J. Med. Chem.* **1975**, *18*, 539.
(6) Free, S. M., Jr.; Wilson, J. M. *J. Med. Chem.* **1964**, *7*, 395.
(7) Hodes, L.; Hazard, G. F.; Geran, R. I.; Richman, S. J. *J. Med. Chem.* **1977**, *20*, 496.
(8) Franke, R.; Huebel, S.; Streich, W. J. *EHP, Environ. Health Perspect.* **1985**, *61*, 239.
(9) Carhart, R. E.; Smith, D. H.; Venkataraghavan, R. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 64.
(10) Grossman, S. C.; Dzonova, B. J.-B.; Randic, M. *Int. J. Quantum Chem., Quantum Biol. Symp.* **1986**, *12*, 123.
(11) Raychaudhury, C.; Basak, S. C.; Roy, A. B.; Ghosh, J. J. *Indian Drugs* **1980**, *18*, 97.
(12) Ray, S. K.; Basak, S. C.; Raychaudhury, C.; Roy, A. B.; Ghosh, J. J. *Arzneim.-Forsch./Drug Res.* **1983**, *33(1)*, 501.
(13) Basak, S. C.; Harriss, D. K.; Magnuson, V. R. *J. Pharm. Sci.* **1984**, *73*, 429.
(14) Roy, A. B.; Raychaudhury, C.; Ghosh, J. J.; Ray, S. K.; Basak, S. C. In *Quantitative Approaches to Drug Design*; Dearden, J. C., Ed.; Elsevier: Amsterdam, 1983; p 75.
(15) Rouvray, D. H. *Acta Pharm. Jugosl.* **1986**, *36*, 239.
(16) Basak, S. C. *Med. Sci. Res.* (in press).
(17) Klopman, G.; Raychaudhury, C. *J. Comput. Chem.* **1988**, *9*, 232.
(18) Klopman, G.; Raychaudhury, C.; Henderson, R. V. *Math. Comput. Modell.* **1988**, *11*, 635.
(19) Harary, F. *Graph Theory*; Addison-Wesley: Reading, MA, 1972.
(20) Deo, N. *Graph Theory with Applications to Engineering and Computer Science*; Prentice-Hall: Englewood Cliffs, NJ, 1974.
(21) Cayley, A. *Philos. Mag.* **1874**, *47*, 444.
(22) Trinajstic, N. *Chemical Graph Theory*; CRC Press: Boca Raton, FL, 1983; Vol. 2, Chapter 4.
(23) Sarkar, R.; Roy, A. B.; Sarkar, P. K. *Math. Biosci.* **1978**, *39*, 299.
(24) Basak, S. C.; Roy, A. B.; Ghosh, J. J. *Proceedings of the Second International Conference on Mathematical Modelling*; University of Missouri: Rolla, 1979; Vol. 2, p 851.
(25) Raychaudhury, C.; Ghosh, J. J. *Proceedings of the Third Annual Conference of the Indian Society for Theory of Probability and its Applications*; Aug. 22–24, 1981; Wiley Eastern Limited: New Delhi, 1984.
(26) Raychaudhury, C.; Ray, S. K.; Ghosh, J. J.; Roy, A. B.; Basak, S. C. *J. Comput. Chem.* **1984**, *5*, 581.
(27) Basak, S. C.; Magnuson, V. R.; Niemi, G. J.; Regal, R. R. *Discrete Applied Mathematics* (in press).
(28) Johnson, M. In *Graph Theory with Applications to Algorithms and Computer Science*; Alavi, Y., Chartrand, G., Lesniak, L., Lick, D. R., Wall, C. E., Eds.; Wiley-Interscience: New York, 1985; p 457.
(29) Andrews, P. R.; Mark, L. C.; Winkler, D. A.; Jones, G. P. *J. Med. Chem.* **1983**, *26*, 1223.
(30) Raychaudhury, C. Ph.D. Thesis, Jadavpur University, Calcutta, India, 1983.
(31) Shannon, C.; Weaver, W. *Mathematical Theory of Communication*; University of Illinois Press: Urbana, 1949.
(32) Klopman, G.; McGonigal, M. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 48.
(33) Dipple, A. *ACS Monogr.* **1984**, *182*, 41.
(34) Camerman, A.; Camerman, N. *Acta Crystallogr.* **1981**, *B37*, 1677.

Use of Vector Processing To Search the Cambridge Structural Database

A. H. M. THIERS and J. H. NOORDIK*

CAOS/CAMM Center, Faculty of Science, University of Nijmegen, Toernooiveld, 6525 ED Nijmegen, The Netherlands

J. BOERHOUT

Convex Computer, Europalaan 514, 3526 KS Utrecht, The Netherlands

Received June 7, 1989

The Cambridge Structural Database (CSD) is a vast numerical resource of crystallographic data. The January 1989 release contains over 70 000 entries, and the data acquisition rate currently increases about 15% per annum. To be able to provide adequate response times for interactive data retrieval, using the new (1988) CSD file format, a vectorized search procedure has been developed as a modification of the CSD program QUEST. This procedure employs the pipelined vector facilities of the CONVEX C120 system to perform bitscreen logic, resulting in response times for arbitrary queries in the order of seconds, almost independent of the size of the database.

INTRODUCTION

The Cambridge Structural Database[1] (CSD) represents a rapidly increasing reservoir of coordinate-based information on molecular structures. This reservoir is accessible via search queries, composed primarily in chemical terms, and in different places[1-3] innovative use of modern computer systems and network facilities helps to deliver this information to the desk of the research chemist. Until 1988 the release files of CSD consisted of three separate files: a BIBliographic file, a CONNectivity file, and a DATA file. Only the former two were searchable with Cambridge Crystallographic Datacentre software, which originated in the 1960s. The computational slowness of this system, combined with more sophisticated user needs, necessitated a major software development effort which resulted in the release, by the Cambridge Crystallographic Datacentre, of the CSD version 3 system in 1988 and the CSD version 4 system in 1989. The main differences with the previous system were a new unified release file and the use of bitscreens to rapidly select candidate structures for exact matches. In the bitscreens a large variety of structural, experimental, and reliability flags are set, and screen out percentages of 95% and higher are easily reached on the more common queries. This screening procedure, combined with direct access file reading, resulted in a typical CPU time requirement of about 200 s on a VAX 11/785 system for a common connectivity query against the 1989 file. On heavily loaded computer systems, this CPU requirement can make on-line searching rather impractical, and this situation will only

0095-2338/90/1630-0019$02.50/0

become worse in view of the rapidly expanding data file size. In this paper we introduce the use of a serial search procedure, using bitscreens on a vector processor, which speeds up CSD searches on a CONVEX C120 system by a factor of about 20 as compared with the VAX version, while maintaining the basic philosophy and the file structure of the CSD file and the QUEST program.

CSD DATA RETRIEVAL METHODS

To search the CSD database interactively, two different approaches[4] have been followed, and implementations for on-line retrieval have been based on either of these approaches.[1] The original version of the pre-1988 CSD system basically used serial searching with string matching for the bibliographic data (author, journal, compound, etc.) and atom-by-atom matching for the (sub)structure searches. Inverted file techniques, where keyed indexes are used to speed up the retrieval process, have also been used. As an alternative to this latter approach, which is much faster than serial searching, the Cambridge Crystallographic Datacentre has chosen the technique of serial searching, applying bitscreens, for its new (post-1988) file structure and software product QUEST. Where both techniques have their specific advantages and disadvantages with respect to database maintenance, disk space requirement, retrieval efficiency, and search speed, only the serial searching technique is open to an additonal increase in search speed, when one uses modern pipelined vector processing hardware (and appropriate algorithms).

VECTORIZATION

Pipelined vector processors[5] have been developed to speed up the floating-point number crunching capacity of scalar computers, but one can employ the specific hardware characteristics of such a machine in other areas as well. In particular, (logical) operations requiring the manipulation of large data arrays can benefit from the ability of a vector processor, like the CONVEX, to manipulate the data in pipelined functional units, with some powerful vector instructions. The compiler can translate loops, the only construct in high-level languages like FORTRAN to manipulate vectors, in one or more vector instructions, thus eliminating loop overhead and speeding up the data processing by a factor of at least 10, as compared with scalar processing. Because the search query for a serial CSD search is composed in such a way that a single pass through the (bitscreen) file is sufficient to select the specified entries, contrary to the inverted file approach, serial searching in principle opens the possibility to vectorize this search loop, which in turn opens the possibility to use the pipelined logic hardware of the CONVEX. It (only) requires a search algorithm from which all constructs hampering vectorization have been removed. This idea has been realized in some new code, which is essentially a modification of the main search loop in the CSD program QUEST.

With the number of entries in the database being N, this (original) search loop is of the form

```
DO I = 1, N
  compare query SCREEN with SCREEN for entry I
    IF SCREENS match THEN
      do additional tests on entry I
    ENDIF
ENDDO
```

The program line "do additional tests on entry I" contains calls to subroutines outside the DO loop. Because the compiler cannot know if variables within the DO loop probably can be changed by the operations outside the loop, this code cannot be vectorized, and it is this piece of code that has been changed in such a way that the "do additional tests" line is moved outside the loop, resulting in an algorithm of the form

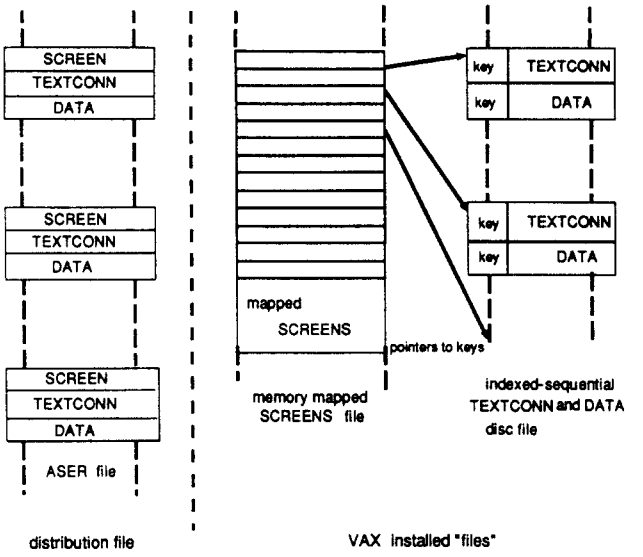

Figure 1. Implementation of Cambridge CSD/QUEST system on a VAX computer, using memory mapping[7] and indexed-sequential data storage.

```
J = 0
  DO I = 1, N
    compare query SCREEN with SCREEN for entry I
      IF SCREENS match THEN
        J = J + 1
        POSSHIT (J) = I
      ENDIF
  ENDDO
DO K = 1, J
    do additional tests on entry POSSHIT (K)
ENDDO
```

where POSSHIT is an array with entry numbers of entries that passed the SCREENS test. This algorithm can be (and is) vectorized by the compiler, and the speed-gain, as a result of the vectorization, is in the I loop, in which basically a FORTRAN IAND operation is performed over all database entries, the bitscreen comparison. When the screening efficiency is high—in practice >95%—the result of this program modification is a very significant reduction in the total CPU time on a pipelined vector processor.

IMPLEMENTATION

In the CSD version 3 system the unified binary search file (ASER) contains SCREEN, TEXTCONN, and DATA records for each entry.[6] The TEXTCONN and DATA records contain structural (connectivity) and atom coordinate information; the SCREEN records are fixed-length 78-integer records containing the bitscreens (a 1 or a 0 indicating the presence or absence of particular information items) in which bibliographic and chemical structure information is flagged.

The original Cambridge implementation of this file on a VAX system, to be used with the Cambridge Datacentre QUEST program, is illustrated in Figure 1. The separate storage of the SCREENS and the TEXTCONN and DATA records improves the retrieval efficiency significantly. The CPU intensive sequential search through the SCREENS results only for hits (matched bitscreens) in pointers to the TEXTCONN and DATA file. In the VAX implementation, these pointers are used for (fast) data retrieval and, in case of connectivity searches, for atom-by-atom matching.

In our CONVEX implementation[8] of CSD/QUEST, applying the vectorized search algorithm as described above, the separation of the SCREENS, and the TEXTCONN and DATA records, as used in the VAX implementation, has been maintained. The vectorized search generates an array

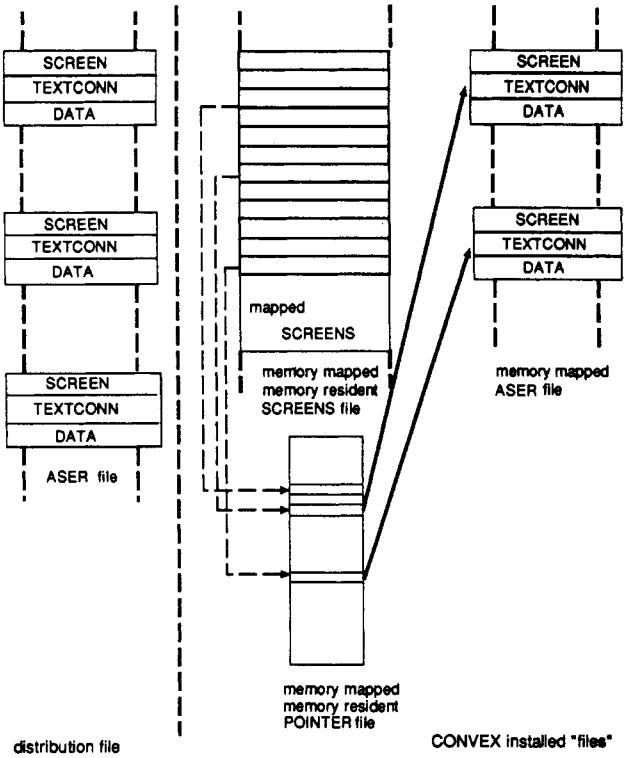

Figure 2. Implementation of modified CSD/QUEST system on a CONVEX computer, using a vectorized bitscreen search with memory-mapped[9] and memory-resident SCREENS and POINTER files and a memory-mapped A SER file.

(POSSHIT) with entry numbers of possible hits, i.e., entries which did pass the screen test and need additional (atom-by atom) testing outside the main search loop. This implementation is illustrated in Figure 2. At load time three files, SCREENS, ASER, and POINTER, a file containing pointers to ASER file entries, are "memory mapped". The additional POINTER file mimics the VAX indexed sequential TEXTCONN and DATA file reading. The memory mapping is performed by the CONVEX UNIX System Call MMAP, which maps a regular file onto the process' virtual address space. The MAP-SHARED option causes the mapped areas to be shared by all processes which refer to the mapped files. The SCREENS file and the POINTER file are generated at load time from the ASER file by a C program and kept memory resident (virtual address space). Although not essential, performing this file generation task only at load time speeds up the startup phase of the search procedure.

Elaborate retrieval runs against different versions of the 1989 release of CSD, ranging from one-sixth of the file up to the complete file, have shown that typical queries are executed within 10 s of (CONVEX) CPU time, independent of the complexity of the query and virtually independent of the size of the database. This last finding indicates that in the optimized program a significant part of the search time is spent outside our vectorized SCREENS loop. Further program optimization might lead to an additional speedup. To use the CONVEX system as efficiently as possible, the hardware configuration at the CAOS/CAMM Center is such that a VAX computer is used as a front-end, handling all interactive processing. Communication between VAX and CONVEX proceeds through CONVEX COVUE (Convex Vax User Environment) software. In the case of CSD this setup is used to interactively create user queries at the VAX and to interactively view the hits on this machine. Only the really time-consuming search procedure is performed at the CONVEX, running the (modified) QUEST program and using the task-to-task communication facilities of the COVUEnet/DECnet software. This setup is illustrated in Figure 3. The

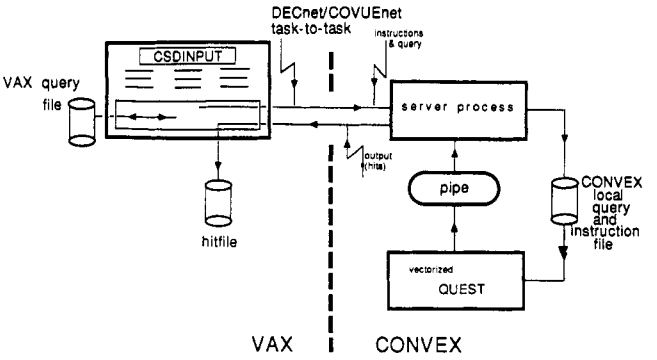

Figure 3. Setup of VAX/CONVEX communication through COVUEnet/DECnet task-to-task communication. Menu-driven query input and hit display program on VAX; modified QUEST search program on CONVEX.

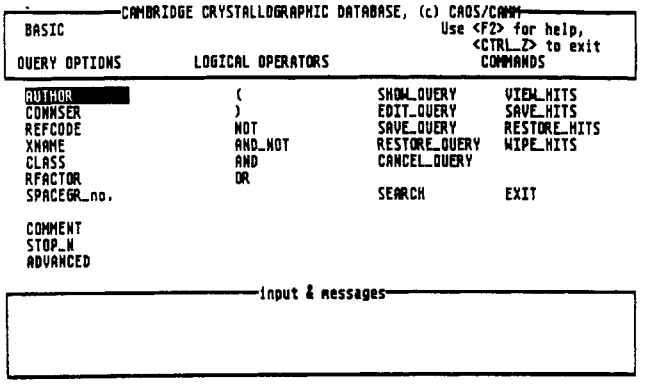

Figure 4. CAOS/CAMM query input menu (program executing on VAX) for CSD. The input and messages box displays user responses to prompts and results from the actual retrieval at the CONVEX.

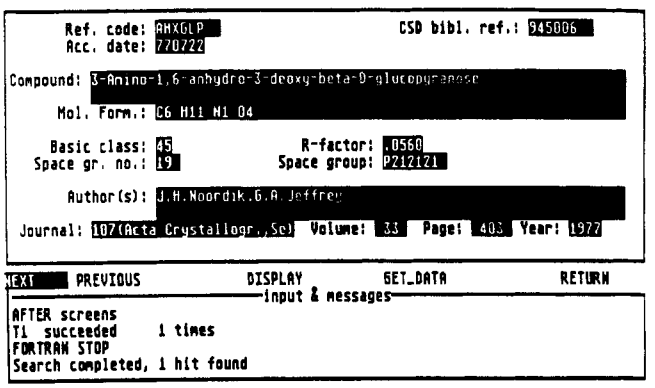

Figure 5. CAOS/CAMM hit display. Display program executes on VAX.

query program starts a server process at the CONVEX which accepts the query plus instructions from the VAX and redirects the output (hits) back to the VAX.

USER INTERFACING

The user interfacing to QUEST to set up the queries (on the VAX front-end) is illustrated in Figures 4 and 5. The user can select the query items (text and/or connectivity) from a menu which is generated by using VAX specific routines. The query input program handles and checks the query syntax. Program output is also displayed interactively at the VAX front-end, and coupling to modeling programs is provided.

Basically the actual search and the query construction and results display are independent, and one could run a search with the CONVEX optimized QUEST software, using only a CONVEX system, but this requires a local (at the CONVEX) query construction option, e.g., an editor and several of the added features of the query input program such as the con-

struction of (sub)structures on a graphics screen are lost.

REFERENCES AND NOTES

(1) Allen, F. H.; Kennard, O. Crystallographic Databases. In *Cambridge Structural Database*; Data Commission of the International Union of Crystallography: Utrecht, 1987; Chapter 2.1, pp 32-76. See also references cited therein. Bellard, S. *Ibid.*
(2) Wood, G. H.; Rodgers, J. R.; Gough, S. R. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 118-123.
(3) CAOS/CAMM Center. Dutch National Facility for Computer Aided Chemistry.
(4) Machin, P. A. *Crystallographic Computing 3*; Sheldrick, C. G. M., Kruger, C., Goddard, R., Eds.; Oxford University Press: Oxford, U.K., 1985; p 106.
(5) *Parallel Computers 2*; Hockney, Jesshope, Eds.; 1988. ISBN 0-85274-811-6.
(6) Cambridge Structural Database System, Users Manual; Version 3.4, Part III, 1989.
(7) VAX/VMS Software Product Description, Release 5.0; Utility SYS$CRMPSC.
(8) Requests for the CONVEX optimized version of CSD/QUEST should be directed to the Cambridge Crystallographic Datacentre, University Chemical Laboratory, Lensfield Road, Cambridge CB2 1EW, U.K.
(9) CONVEX UNIX Programmers Manual, Part II, Section 2.

Prolog-Based Functional Group Perception and Calculation of 1-Octanol/Water Partition Coefficients Using Rekker's Fragment Method

KEI TAKEUCHI, CHIAKI KURODA, and MASARU ISHIDA*

Research Laboratory of Resources Utilization, Tokyo Institute of Technology, 4259 Nagatsuta, Midori-ku, Yokohama, Japan 227

Received June 15, 1989

A new algorithm is developed to perceive functional groups in a chemical structure from the method that generates simple pairs of atoms comprising functional groups. Description of the program to perceive functional groups is given. The program is applied to calculate log *P*, logarithmic 1-octanol/water partition coefficients, on the basis of the method developed by Rekker in which log *P* is calculated from fragment constants for various functional groups.

INTRODUCTION

Perception of functional groups is essential for the computerized chemical structure-handling techniques[1] in structure–activity correlation, physicochemical property estimation, and computer-assisted organic synthesis design.

There are four ways to automatically perceive functional groups. The first one, the most time-consuming one, is to perceive a list of functional groups of interest by atom-by-atom search. The second one, which is more sophisticated in searching a list of functional groups, is the "table-driven" method,[2] which involves logical manipulation of the list. An algorithmic definition of functional groups is the third approach. Chu et al.[3,4] used augmented atom fragment[5] to satisfactorily represent functional groups in substructure analysis. Although this method is much faster than the atom-by-atom search, problems of redundancy caused by overlap of structures arise.[1] The fourth is an algorithmic perception method using an encoding scheme that uniquely represents each functional group. This method has been developed in the organic synthesis design[6,7] and the mechanistic evaluation of organic reactions.[8] A canonical connection table is employed in the former to compile the machine formulas for functional groups from the viewpoint of a central atom of the group, whereas, in the latter, information on where functionality begins and ends or how paths grow from the origin is important for compiling codes for each functional group.

The present paper is concerned with the program to perceive functional groups with logic programming language Prolog.[9] Prolog has been developed as an artificial intelligence language and widely applied to the field of chemical structure handling, such as molecular property estimation,[10] calculation of topological index,[11,12] organic synthesis design,[13,14] and structure elucidation.[15-18] These applications demonstrate that Prolog is useful not only for a knowledge-based system but also for manipulation of chemical structures. Prolog programming has several advantages. Substructure search becomes quite simple by using the inference engine installed in Prolog.[12,17] The algorithm is so clear that the program becomes very compact and is easily developed.

When the routine to perceive functional groups becomes available, one can apply it to calculation of log *P*, the logarithm of the 1-octanol/water partition coefficient. log *P* is a well-known hydrophobic parameter that is closely correlated with various biochemical activities. This value is hence widely used for a descriptor in structure–activity studies, and its estimation is proposed by Rekker.[19,20] Rekker's method is based on the assumption that this value possesses additive–constitutive character. Rekker assigned "hydrophobic fragment constants" to each substructure, and the log *P* value can be calculated by appropriate fragment constants and correction factors. The fragments consist of functional groups and carbon atoms with attached hydrogen. In that calculation, Rekker used several correction factors that are also related to the interactions between the functional groups. Hence, a powerful routine to perceive functional groups must be developed to calculate log *P* by Rekker's method. Recently, Darvas et al.[10] developed a calculation system for Rekker's method programming in Prolog, called PRO-LOGP. However, the algorithm to perceive functional groups in it is not reported.

PERCEPTION OF FUNCTIONAL GROUPS

Functional groups as well as ring fragments are chemical and problem-dependent substructures. Their constituent atoms are generally heteroatoms, such as S, N, O, and P, and halogens or unsaturated carbon. In some cases, hydrogen atoms are attached to them. The size and shape of their skeletal structures are not fixed. This causes difficulty in perceiving functional groups by a computer.

An efficient algorithm without encoding scheme, which we call "simple pair generation method", is outlined here. A simple pair X–Y comprises a bond by which two atoms X and Y are linked. Suppose that functional groups are classified into elementary and complex ones. Functional groups such

0095-2338/90/1630-0022$02.50/0