time for a graph of size *n*. If a chemical structure of size *n* is transformed into a usual graph and the algorithm is applied, the total processing time becomes

$$c^k a_k n^k + c^{k-1} a_{k-1} n^{k-1} + ... + a_0 + d_1 n + d_0 + b$$

where *b* denotes time required for subroutine call.

However, as the unique naming algorithm for the graphs of bounded valence is too complicated to implement (it is based on the group theory) and the degree of the polynomial is large, it is not considered to be practical. For almost all inputs occurred in practice, such algorithms as the Morgan algorithm seem to work much more efficiently.

What we want to point out here is that there is a possibility that more practical and efficient algorithms for unique naming of graphs will be found. Once such an algorithm is found, it can be applied to stereochemically unique naming by using the results of this paper.

The transformation method can be applied not only to unique naming but also to substructure matching, because transformed graphs are locally ismorphic if and only if original chemical structures are locally and stereochemically isomorphic. Unfortunately, no polynomial time algorithm is known for subgraph matching even if graphs are restricted to the ones of bounded valence.[5] However, once an efficient algorithm for subgraph matching is found, it can be directly applied to stereochemically substructure matching.

## CONCLUSION

A method which transforms stereochemical structures into graphs (structures which do not have stereochemical information) is presented. The transformation is very simple, and two structures are transformed into isomorphic graphs, if and only if they are stereochemically isomorphic. By using the method, graph algorithms for usual graphs such as unique naming algorithms and subgraph matching algorithms can be applied to stereochemical structures. That is, when an efficient algorithm for usual graphs is found, it can be applied to stereochemical structures. A polynomial time algorithm for stereochemically unique naming is implied as an example.

ACKNOWLEDGMENT

I thank Prof. Ohsuga of the University of Tokyo for giving me a chance to study computer application to chemistry. Also, I would like to express my gratitude to Prof. Sasaki and Dr. Funatsu of Toyohashi University of Technology for their valuable discussions and suggestions on chemical information processing.

## REFERENCES AND NOTES

(1) Aho, A. V.; Hopcroft, J. E.; Ullman, J. D. *The Design and Analysis of Computer Algorithms*; Addison-Wesley: Reading, MA, 1974.
(2) Akutsu, T.; Suzuki, E.; Ohsuga, S. A. Logic Based Approach to Expert Systems in Chemistry. *Knowl. Based Sys.* (in preparation).
(3) Dubois, J. E. French National Policy for Chemical Information and the DRAC System as a Potential Tool of This Policy. *J. Chem. Doc.* **1973**, *13*, 8–13.
(4) Furer, M.; Schnyder, W.; Specker, E. Normal Forms for Trivalent Graphs and Graphs of Bounded Valence. *Proc. ACM Symp. Theor. Comput.* **1983**, *No. 15*, 161–170.
(5) Garey, M. R.; Johnson, D. S. *Computers and Intractability*; Freeman: San Francisco, 1979.
(6) Hendrickson, J. B.; Toczko, A. G. Unique Numbering and Cataloguing of Melecular Structures. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 171–177.
(7) Kudo, Y.; Sasaki, S. Principle of Exhaustive Enumeration of Unique Structures Consistent with Structural Information. *J. Chem. Inf. Comput. Sci.* **1976**, *13*, 43–49.
(8) Luks, E.M. Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time. *Proc. IEEE Symp. Foundat. Comput. Sci.* **1980**, *No. 21*, 42–49.
(9) Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures—A Techenique Developed at Chemical Abstracts Service. *J. Chem. Doc.* **1965**, *5*, 107–113.
(10) Petrarca, A. E.; Lynch, M. F.; Rush, J. E. A Method for Generating Unique Computer Structural Representation of Stereoisomers. *J. Chem. Doc.* **1967**, *7*, 154–165.
(11) Randic, M. On Canonical Numbering of Atoms in a Molecule and Graph Isomorphism. *J. Chem. Inf. Comput. Sci.* **1977**, *17*, 171–180.
(12) Sussenguth, E. H. A Graph-Theoretic Algorithm for Matching Chemical Structures. *J. Chem. Doc.* **1965**, *6*, 36–43.
(13) Wipke, W. T.; Dyott, T. M. Simulation and Evaluation of Chemical Synthesis—Computer Representation and Manipulation of Stereochemistry. *J. Am. Chem. Soc.* **1974**, *96*, 4825–4834.
(14) Wipke, W. T.; Dyott, T. M. Stereochemically Unique Naming Algorithm. *J. Am. Chem. Soc.* **1974**, *96*, 4834–4842.

# Compact Numeric Alkane Codes Derived from IUPAC Nomenclature

SCOTT DAVIDSON*

Computer Data Systems, Inc., One Curie Court, Rockville, Maryland 20850

Received January 18, 1991

A reversible binary coding scheme for storing and ordering all alkane isomers through $C_{21}$ as 32-bit integers is described. The method is derived from a modified set of IUPAC rules formerly utilized to cite side-chain names by increasing complexity (Davidson, S. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 151–5). The ordering enables construction of a bitwise tiebreaker series in which the number of bits assigned at each step is determined by the remaining choices. The compactness of the resulting codes compares favorably with previously reported graph-based codes. Manual encoding/decoding is not difficult because bit fields are small, and the logic is based upon already familiar considerations of chain sizes, lengths, and locants.

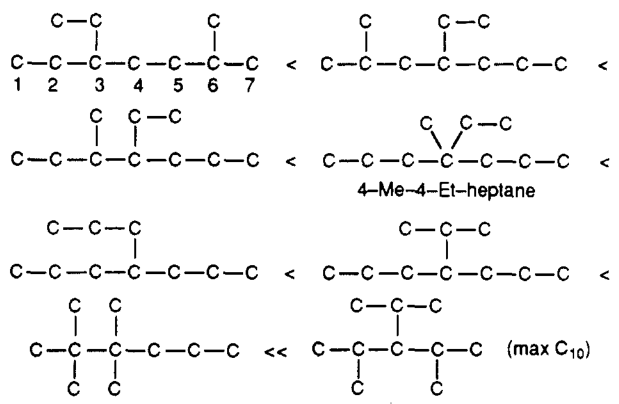
## INTRODUCTION

Many numeric codes for alkanes have been reported in the literature. However, only a few have proved to be both unique and reversible. Gordon and Kennedy[1] developed a compact, ordered integer code from combinatorial equations related to enumeration of rooted trees, ordering alkanes by increasing chain length. Decoding is obtained by iterative solution of the same equations. Knop et al.[2] designed an "N-tuple" code for alkanes consisting of a string of *N* digits for an *N*-carbon alkane. The string represents a maximal sequence of the number of uncounted bonds at each carbon in a traverse of the longest unexplored path from a most substituted carbon, then backtracking to visit all other carbons in that path. Randić[3] has extended this code to polycyclic structures and has reviewed other codes for comparison.

A numeric code derived from standard nomenclature and ordered by size would have the advantage of being a more

* Address correspondence to 240 Manor Circle No. 2, Takoma Park, MD 20912.

Chart I



familiar point of departure for chemists. This paper describes such a coding system, one that allows for either manual or computer storage and retrieval of any of the first million alkanes by size (through $C_{21}$) as a 32-bit signed integer. Simple integer sorting orders alkanes by size, by length, and by a logical side-chain ordering.

While it is not surprising that most numeric alkane codes have been derived from graph theory, it would still seem that the IUPAC rules could offer additional opportunities. Perhaps a perceived lack of ordering has deterred effort in this area. Read[4] cites the main-chain tie problem in this regard. However, both ordering and tie-handling problems are overcome by a side-chain complexity minimization algorithm described in a previous paper.[5] Two basic modifications to the IUPAC rules[6] are as follows: (1) The main chain is defined as "the chain with the least complex side chains" and (2)—as a consequence of (1)—the most complex (largest) side chain determines the direction of main-chain numbering. Both imply the use of a tiebreaker series when necessary. In many cases (particularly methyl-only alkanes) the results obtained agree with the IUPAC rules. However, when an alkane has more than one longest chain, the algorithm was shown to give consistent results when breaking ties. Specifically, four sequential tiebreaker rules that do not always function as intended are replaced by a simple recursive algorithm. No new candidate main chain is compared with the current best until all of its side chains have been completely characterized. The selection order (size, length, locant) utilized to determine relative side-chain complexity is based upon a set of former IUPAC rules[7] for citing side chains in an alkane name—after the main chain had been selected by other rules. Complexity ordering embodies a living-tree concept, in that the lowest complexity side chains of equal size and length are those whose largest branches are closest to the main chain (trunk).

Any alkane database ordered as unique numeric codes can be updated quickly by computer. However, the ordering itself may not be readily related to the structure, i.e., it may be very difficult to determine by inspection which entries would precede or follow a given entry in the database. With complexity ordering, the logic is easy to follow, as shown in Chart I with the three nearest neighbors of 4-methyl-4-ethylheptane of both lower and higher complexity. Notice the complexity-ordered name and also the non-IUPAC numbering of the first skeleton, where the ethyl group takes the low locant. (Continuing to the left, the methyl locant decreases until the highest complexity trimethylheptane is encountered—the 2,4,6 isomer.)

## ALGORITHM PROPERTIES

The coding method is based upon compression of an "alkane specification page"—a set of one or more lines of side-chain codes ordered by decreasing complexity. The top line codes the entire alkane taken as a side chain, while lower lines—if any—code the same type of information for complex side chains ($>C_4$). (See Figure 2 for page details.) In Scheme I, the top line shows the relationship between a carbon skeleton and its standard alphabetic and numeric descriptions. Addition of the specification page, generated from a simple recursive algorithm, creates a facile link between an alphabetic alkane name and its numeric compact code. Translating or compressing the page each requires fewer than 200 lines of computer code. While generation of the page itself requires about twice as much code, in most cases this can be done manually by inspection for direct input. Since a page consists of subpages, the top line that codes a complete alkane is indistinguishable from a complex side-chain line. Therefore, the same list–sort procedure that orders the complex side chains of an alkane can also be utilized to order the alkanes in a database, as indicated in the diagram. Individual alkane pages are concatenated to give a single large page, and their lines are renumbered (logically) by adding a single top line giving the beginning line numbers of each page (2, ⟨P2L1⟩, etc.). Sorting this line orders the database.

Scheme I

$$\begin{tikzcd}
\text{IUPAC name} \arrow[d, "\text{translate}"', leftarrow, phantom] & \text{C-skeleton} \arrow[l, "\text{rules}"'] \arrow[r] \arrow[dl, "\text{recursive algorithm, list-sort}"'] & \text{connection table} \arrow[dr] & \\
\text{specification page} \arrow[rr, "\text{compress}"] \arrow[d, "\text{translate}"] \arrow[dr, "\text{(several) list-sort}"] & & \text{32-bit integer code} & \text{numeric code (s)} \arrow[d, "\text{(several) sort}"] \\
\text{(mod.) IUPAC name} & \text{ordered page database} \arrow[rr, "\text{compress}"] & & \text{ordered integer database}
\end{tikzcd}$$

Unique numbering of atoms permits extension of alkane codes to include heteroatoms and multiple bonding. Morgan's algorithm[8] for generating unique numberings from connection tables (as optimal successor lists) became the basis of the CAS Registry System. Nodal nomenclature[9] gives unique numbering of an alkane as a minimal alternating sequence of longest remaining unnumbered chain lengths and their locants on the parent chain. Side-chain complexity minimization generates unique numbering similarly, but in a depth-first fashion with size as the high-level qualifier. A simple analogy that *contrasts* these two methods is useful: Nodal nomenclature would describe a living tree as successively shorter branches, beginning with the trunk, whereas complexity ordering sees the same tree as smaller trees made up of smaller trees, etc., beginning with the whole tree. Thus in reducing a downed tree to branches, the largest (heaviest?) branch would be completely cut up—down to the smallest twig—before the second largest would be touched. (A "nodal woodcutter" would put in a lot more legwork!)

Global unique numbering may prove useful for extending this coding system but is not essential. For example, substitution of carbon by oxygen to give an ether linkage can be specified by local numbering within a substructure. Global numbering is immediately available by adding the number of atoms already identified.

## CODING METHOD

The tabular specification-page format is convenient for programming. However, although locants have been combined with side-chain codes, storage is still inefficient for large alkane database use, and sorting requires a special list–compare routine for the complex side chains which is slower than simple sequential compares. Computing quite often presents conflicts between speed and storage. However, in this case, if the page can be compressed to a single value while preserving the linked ordering, then gains are made in both.

This paper describes a simple but careful binary coding scheme that utilizes specification-page information in stepwise assignment of bits, based upon diminishing choices for the remaining side-chain parameters at each step. The coding logic is based upon considerations of substructure size, chain lengths, and locants which are already familiar to most chemists. Since nearly all bit fields are limited to widths of four or less, manual encoding/decoding is readily learned with the aid of a small binary-decimal conversion table and several practice runs. The decoding process is similar to that of a simple change-making program, where a given sum equals the complete alkane; dollar bills equal complex side chains; quarters, dimes, and nickels equal butyl, propyl, and ethyl; and pennies equal methyl groups (the remainder).

## SIZE/LENGTH FORMATS

The complete alkane code is stored left-justified as a 32-bit signed integer. The size format is designed to provide more storage for the largest sizes. The first two bits are coded as follows: 11 = $C_{21}$; 10 = $C_{20}$; 01 = $C_{19}$; 00 = $<C_{19}$. If the leftmost (sign) bit is 1, then the integer is negative and any $C_{20}$ or $C_{21}$ alkanes will be ordered as negative numbers. If either of the first two bits is a 1, then the next four bits give the chain length in complement form—the number of side-chain carbons ($C_s$). This format is necessary to maintain a continuous low compare when sorting alkanes by complexity. More importantly, this number serves as an initial value that *decreases* as side chains are identified, thereby reducing bit-storage requirements for those remaining. Decoding can be described as a simple loop (do while $C_s > 0$) wherein the coded right half of a double word (0, ⟨code⟩) is shifted left by a computed number of bits and the left half is evaluated as an integer to obtain structural information. (Detailed pseudocode is given in the appendix.)

The complete size/length format is shown, with brackets indicating use conditional upon the presence of leading zeros:

```
ss   [ssss]  [ss]  1111
19+  4–18    1–3   0–15
```

For database use, this demonstration format would be modified to increase the size limitation, and the rightmost bit would be reserved as a flag to indicate completion or continuation of coding—possibly in the next 32-bit word. Coding of side chains is the most important factor in efficient storage and is best illustrated by example in what follows.

## NUMEROUS METHYL, ETHYL COMBINATIONS

Any code must be able to handle the large number of methyl isomer combinations possible. For a given alkane size, storage requirements should reflect the number of isomers that must be distinguished. The problem is to devise an efficient method for handling the most difficult cases. Table I shows more than 8000 isomers for $Me_7$ tetradecanes. If two Me groups are replaced by ethyl (locant 3–7), the reflective symmetry which otherwise creates high-locant duplicates is destroyed and there are more than 15 000 isomers.

To free more storage space for the numerous combinations, a single bit follows the side-chain carbons field (if $C_s > 2$) to indicate the presence of side chains larger than ethyl. (This is the third tiebreak or separation point during sorting.) If not set (0), then all side chains are methyl and ethyl, and the remaining bits become available for the details. In such case, 1–3 bits are allocated for the ethyl count, which can be 0–7 depending on $C_s$ (3–15). (Actually, the maximum is 6 with the $C_{21}$ size limit—hexaethyl heptane.) This is the fourth tiebreak point for Me/Et-only isomers with matching bits to here; fewer ethyl groups mean a less complex isomer with more Me groups. A non-zero ethyl count is followed by one or more bit fields giving locants from $L$-2 (down) to 3, where $L$ is the main-chain length.

Table I. $C_{21}$ Methyl Alkane Isomers

| length | Me groups | isomers |
|---|---|---|
| 20 | 1 | 9 |
| 19 | 2 | 81 |
| 18 | 3 | 400 |
| 17 | 4 | 1435 |
| 16 | 5 | 3549 |
| 15 | 6 | 6399 |
| 14 | 7 | 8118 |
| 13 | 8 | 7215 |
| 12 | 9 | 4175 |
| 11 | 10 | 1471 |
| 10 | 11 | 252 |
| 9 | 12 | 16 |
| | | total 33120 |

The number of methyl groups need not be stored, since this is simply the number of carbons remaining after all other side chains have been found. There are two basic ways to locate methyl groups ($L$-1 to 2):

(1) by locant of individual methyl groups

(2) by the number of methyl groups (0–2) on each locant position

For fixed total carbons (Table I), method 1 is more efficient for a long chain with a few Me groups, while method 2 is better for a highly substituted short chain. For the $Me_7$ tetradecane isomers, four bits (2–17) are required to store any locant up to 13 using method 1. Thus a fixed format would require 28 bits just to store the Me groups. However, the descending complexity ordering enables a more efficient coding based upon remaining choices—the methyl locants are numbered in reverse. While four bits are required initially, once a locant of 9 or less appears, three bits (2–9) suffice, etc. In general terms, leading zeros in a same-type coding set (e.g., Me groups) can be dropped following their first appearance. (An obvious restriction that reduces storage requirements for alkanes having only identical side chains is that the given numbering be correct. Thus a 13-Me tetradecane must have at least one 2-Me group, a 12-Me, a 2- or 3-Me, etc.)

For method 2, two bits (0–3) are required for each position coded from $L$-1 to 3 (any remaining are 2-Me groups). The fourth value (3) is unused. Compression to a ternary format would save about one bit in five [log 3/2 log 2 = 0.792]. For example, the methyl count of five locant positions requires 10 bits, but could be compressed into an 8-bit byte (0–255; $3^5 - 1 = 242$). However, an improved binary code that also extends the range of efficient use to longer chains is obtained by using two zero codes as shown below. Locants ($l$) for the next two Me groups are given to show that the zero codes also produce the correct advance tiebreak necessary to avoid "apples and oranges" comparisons in the next step during sorting of isomers identical to this point.

11 = 2 Me groups ($l$,$l$); 10 = 1 Me group ($l$,$<l$)

01 = none here but 1 or 2 on the next lower locant ($<l$,$<l$)

00 – none here and also none on the next ($<l-1$,$<l-1$)

Following 01 only one bit is needed to indicate one or two Me groups. Code 00 indicates that the next two bits apply to two positions down the chain. Sparse chains can be traversed quickly by skipping every other vacant position. Still, at some point—as chain length increases for a given number of Me groups—method 1 will be more efficient. For this work alkanes with three or fewer Me groups use method 1. Also, the last (lowest) locant is always coded individually because this will require fewer bits in the event of a large gap between the last two. (A general and more accurate way to choose between methods is given below.)

**Chart II**

Specification Page:

| line | C(total) | C(side) | side chain codes | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 21 | 7 | 205 | 112 | 109 | 107 | 107 | 103 |

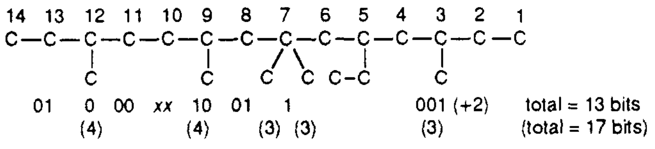
3, 7, 7, 9, 12–pentamethyl–5–ethyltetradecane

Binary code:

size side $Y$ $E_m$ (+3)

|11|0111|0 |01 |010| <5 Me locants>

**Chart III**


14 13 12 11 10 9 8 7 6 5 4 3 2 1
01 0 00 xx 10 01 1 001 (+2) total = 13 bits
(4) (4) (3) (3) (3) (total = 17 bits)

$N$-tuple code (path 7, 8, ...): 412112100012121001000

For the first example, one of the more than 15000 ethyl, pentamethyl tetradecane isomers has been selected. The specification page and binary code are shown in Chart II, followed by a stepwise description of the decoding process. (The corresponding $N$-tuple code is also given.) Simple side chains (taken as $C_1$–$C_4$) use 3-digit codes SLL where S = 1, 2, ..., 8 for methyl, ethyl, ... *tert*-butyl and LL is the locant. The complexity ordered name is obtained by reading the page from right to left.

A $C_{21}$ alkane with seven side-chain carbons ($C_{14}$ main chain) has only methyl and ethyl side chains ($Y$). There can be at most $E_m$ = int(7/2) = three ethyls requiring 2 bits (0–3). Reading the next two bits gives 1, and we compute $C_s = C_s - 2$. Since the single ethyl is most complex, its locant must be 3–7 or 5 possible; uses three bits with offset of 3. The five carbons remaining must be methyl groups. Since there are more than three, method 2 is chosen. The locant position codes are shown in Chart III.

The zero code 01 indicates no Me on C-13, but 1 or 2 on C-12. Code 0 for C-12 says one Me. Code 00 for C-11 means skip C-10; the next two bits apply to C-9 and indicate one Me group. Code 1 on C-7 following the zero code on C-8 says two methyls here. The remaining Me is coded by locant, range 2–7 (or 2–6 if occupied C-7 is noted). Either requires three bits. The complete structure is specified by 25 bits total. Shown below the position codes (parentheses) is the number of bits required to code each Me group individually by locant (method 1). Notice that after the leading zero appears in the code for locant 9 (0111, +2), only three bits are needed.

Because this is a variable length code (beginning with size and ending when all side-chain carbons have been found), it is necessary to consider the worst cases for both methods to compare compactness with codes which have fixed length for alkanes of the same size, such as the $N$-tuple codes of Knop et al.[2] The worst cases for method 1 occur when a larger side chain allows the Me groups to be bunched at the high-locant end, whereas with method 2 they occur when each occupies a different consecutive position increasing from locant 3. Figure 1 shows two isomers of example 1 to illustrate this.

In method 1, Me groups with locants ≥10 require four bits, while in method 2, two bits are used for each singly occupied position and two bits are 'used for every other vacant position. If the bit requirements for the two methods are equated, a useful formula to estimate the minimum locant (1) for switching from method 2 to method 1 as a function of the number of remaining Me groups ($n$) is obtained. Thus for Me groups in the 4-bit uncertainty zone (1 > 9):

$$4n = 2n + 2(1 - 2 - n)/2 \text{ or } 1 = 3n + 2$$

For $n$ = 5, 1 = 17; thus in the examples given, the main chain would have to be lengthened to 18, for a total of 25 carbons, for method 1 to be competitive. However, for fine-tuning, the choice of method could be reevaluated after each Me locant is found or position counted, without damaging the tiebreak sequence.

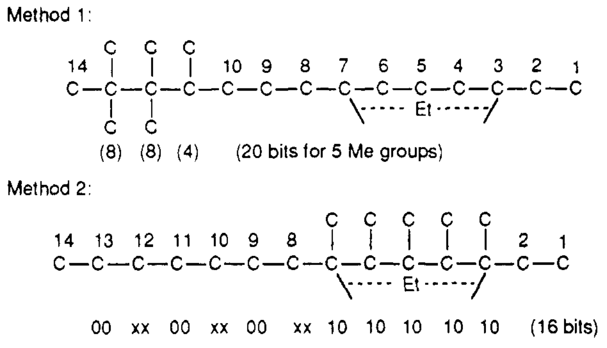

Method 1:
14 10 9 8 7 6 5 4 3 2 1
Et
(8) (8) (4) (20 bits for 5 Me groups)
Method 2:
14 13 12 11 10 9 8 2 1
Et
00 xx 00 xx 00 xx 10 10 10 10 10 (16 bits)

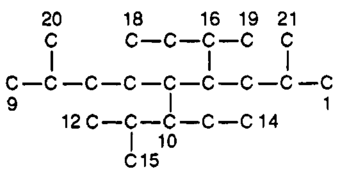
**Figure 1.** Worst-case storage examples for two methods of methyl location.

It would appear that 16 bits is an upper limit for storage of the five methyl groups in the $C_{21}$ isomers above. Together with the 12 bits required for the first part of the code, this totals 28 bits. If an unbiased size code of five bits (1–32) is utilized instead, the total is 31 bits. For comparison, each digit of the $N$-tuple code (1–4 for the first, 0–3 for the rest) is compactly represented by two bits. Since a $C_{21}$ alkane is coded by 21 digits, 42 bits are required. A detailed analysis of the difference is beyond the scope of this paper. However, it would appear that the reduction in uncertainty which accompanies the complexity order decoding process as the number of unidentified side-chain carbons and their possible locants is reduced is a major factor.

Comparison of computer decoding of both codes in binary format presents a classic tradeoff between speed and storage. The shorter complexity code requires computation of the number of bits to be shifted left across a word boundary at each step, whereas the $N$-tuple code is merely shifted a constant two bits for each code value until the sum of ⟨value⟩ − 1 goes negative, ending the code and giving $N$. On the other hand, a complexity code can be compared directly (as a 32-bit integer) with other complexity codes for sorting by size, length, etc. without resorting to decoding since the ordering is built-in during the coding process. Moreover, most known alkanes are much simpler than the worst case examples above. Thus, any trimethyl alkane of size 21 or less can be stored with 20 or fewer bits, leaving 12 low-order bits to store extra information (e.g., a 4-digit boiling point in the range 0–409.5 in an implied F5.1 format). Needless to say, this would have no effect on the sorted ordering.

## COMPLEX, BUTYL AND PROPYL SIDE-CHAIN CODES

Complex side chains, defined as having five or more carbons, are handled generally as second alkanes nested within the first. Restrictions reduce storage needs. For example, complex side chains must have at least five carbons, but no more than the total number of side-chain carbons ($C_s$). Their length must be at least 3 (*tert*-pentyl), but no more than the smaller of about half the main-chain length [int($L$ − 1/2)] and $C_s$. While there are more than four side-chain carbons left, a "complex" bit tells whether the next bit sequence describes a complex side chain. Complex side chains are coded on the page in the 5-digit format 1NNLL, where NN is the line where the side chain is further simplified and LL is the locant, as shown in Figure 2. This page shows a main-chain tie in the complex side chain, where the initially found isopropyl group in line 2 (**) was replaced by line 3. Also shown is the unique numbering generated, which is the same as the order of visit



Specification Page:

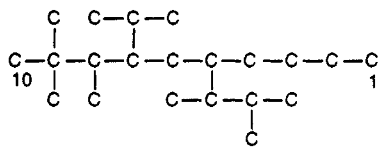
| line | C(total) | C(side) | side chain codes | | | |
|---|---|---|---|---|---|---|
| 1 | 21 | 12 | 10305 | 604 | 108 | 102 |
| ·· | 6 | 3 | 401 | 0 | 0 | 0 |
| 3 | 6 | 3 | 201 | 102 | 0 | 0 |

2, 8–dimethyl-4-*sec*-butyl–5–(2–methyl–1–ethylpropyl)nonane

Binary code: (29 bits)

```
 size side  Y  C   +5  +2  Y  E  +1   +4  C  B  s  +4   E   +2    +2
|11|1100|1|1|(|001|1|0|1|1|)|01|0|1|01|0|0|110|000|
```

**Figure 2.** Complex side-chain coding example.



Binary code: (standard = 35 bits, modified = 32 bits)

```
 size side  Y C  +5 +1 E <+1> +4 C B  P I +3  E <  Me(+2) >
|11|1011|1|1|(000|1|0|1|0|)|01|0|0|01|1|100|0|111|111|110|
              C'  V 1, 2dMP
|11|1011|1|0|01|101          |01|x|0|01|1|100|0|111|111|110|
```

**Figure 3.** Approaching maximum storage for $C_{21}$ isomers.

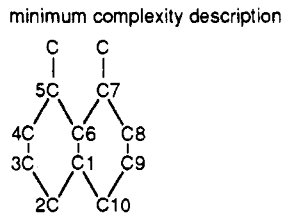
that would produce the optimal page without sorting or replacements.

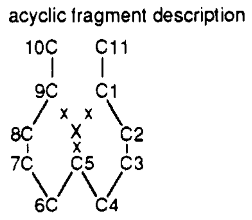
Butyl and propyl codes are both preceded by counts, since a mismatch here breaks ties. Two bits follow for butyl to indicate one of the four types (*sec*-butyl, etc.), and one for propyl to distinguish isopropyl from propyl.

This code shows a $C_{21}$ nonane with $C_s$ = 12, not all Me and Et, and the largest is complex. Inside the parentheses three bits say 6 C's in the range 5–12. The length range is from 3 to int(8/2) = 4. Therefore the number of side-chain carbons must be three or two respectively; only one bit is needed to say three. The next two bits (*Y* and *E*) indicate Me/Et only, one ethyl. There is no choice of locant for the ethyl group so no bits were assigned. With one C left inside, the last bit indicates a 2-methyl. The two bits following the right parenthesis indicate locant 5 (choice 4–6) for the complex side chain. (Locant 6 is valid if only length is considered, because an identical side chain could be on locant 4. However, this would create a new main chain.) With six C's now left, a second complex side chain is possible; the next bit says no. There can be 0 or 1 butyl group (*B*). The code 01 says *sec*-butyl, limited to locants 4 or 5. With only two C's left, there is no ethyl, so the remaining code must be for two Me groups. Because *sec*-butyl determines the chain numbering (the complex group is in the middle), these may be anywhere in the range from 2 to 8, so three bits are required for each. If there had been two complex side chains, the size of the second one would be limited to 5 or 6, and it would either be identical with or less complex than the first one.

The next example (Figure 3) is a near-overall worst case obtained by combining worst-case methyl group situations in both main and complex side chains. This structure breaks the 32-bit field limit, encouraging reconsideration of the arbitrary division between simple and complex side chains.

This code is similar to the preceding one, except the chain length of 10 wrecks the symmetry, allowing the four remaining side chains to take high (3-bit) locants. Following the $C_5$ code, there is no butyl group, so there can be 0–2 propyl groups. The





5, 7–dimethyl[1, 6]cyclodecane

```
Binary code: (total 25 bits)                       N-tuple/Ring closure code:
 size  (+3)  Cs   R B T  +3  B E <Me(+1)>          ←—Cl–C11—→
|00|1001|0010|1|1|0|011|0|0|0110|100|             2 1 1 1 1 1 1 1 1 0 0 3 1 5 9
```

**Figure 4.** Comparison of two codes for a bicyclic structure.

code (*P* and *I*) indicates one isopropyl group on locant 7. Shown below the standard code is a modified code in which simple side chains are extended to include the eight $C_5$ side chains. The decreasing complexity order ranges from 1-ethylpropyl to *n*-pentyl. The side chain in the code (1,2-dimethylpropyl) is the third most complex, following 2,2-dimethylpropyl (neopentyl). The initial complex code (*C'*) now becomes zero since the test was for $>C_5$, and a count code (*V*) is added for $C_5$ as with the $C_4$, $C_3$, and $C_2$ codes. This coding change saves two bits in the $C_5$ code and eliminates a second complex bit, reducing the total to 32. Extending the simple side chains much further would not be very practical because the count codes added for each size would build up, and of course the rapidly increasing number of "simple" side chains would create handling problems.

## RING EXAMPLE

The final example (Figure 4) is given to show that this coding method is not limited to acyclic structures. (A future paper will describe in detail the extension of complexity-based nomenclature and coding to polycyclic systems.) The structure shown is the first example from Randić's 1986 paper[3], utilized to illustrate extension of the *N*-tuple code to cyclic structures. The 15-digit code consists of the *N*-tuple code for the acyclic fragment obtained by erasing the top bridgehead carbon (*X*), to which is appended the number and values of *N*-tuple locants of ring-closure neighbor carbons. The computer-generated name given is nonstandard, but seems closest to nodal nomenclature[9]; the structure is described in terms of a perimeter ring, bridge, and side chains. Complexity ordering of bridges here follows the classical view that a bridge belongs to the main ring framework. Therefore, bridges are more complex than side chains. Again, complexity numbering tries to minimize the highest rather than lowest locants. The low locant of the first (most complex) bridge is set to 1 to minimize the high locant, and so requires no storage. In this case the bridge numbering is standard, but the methyl locants become 5,7 rather than 2,10.

To code the example, three bits are inserted after the 4-bit $C_s$ field; a ring bit (*R*), a bridge bit (*B*), and a bridge-type bit (*T*) to indicate a directly connected bridge (no C's). The high locant is between 3 and 9 which uses three bits. (If recorded in advance that only one bridge is present, then the limits are from three to halfway around the ring, or six.) Next, a second bridge bit set to zero tells us that there are no more bridges, and the acyclic coding resumes with the ethyl-count bit (*E*). The two remaining carbons are not an ethyl group. The high methyl locant can be anywhere (1–10) and requires four bits. The low locant must be ≥7 and thus uses three bits.

## CONCLUSION

The traditional and familiar parameters utilized to describe chain and ring structures *can* be translated into practical compact numeric codes.

## APPENDIX: PSEUDOCODE FOR DECODING ALKANES THROUGH $C_{21}$ SIZE

(Encoder follows the same rules for assigning field widths.) Uppercase letters are used for arrays, functions, and subroutines.

For ease of reading, results variables are underlined, but their storage is omitted. Bit-field reductions based upon prior occupancy of locants, switching methods during location of methyl groups, and eliminating locants that would create new main chains are omitted. Details of the symmetry computation are also omitted. When the main chain is currently symmetric, high/low locant pairing of identical side chains is omitted, but limiting the high locant of a single (next) most complex side chain to the low half of the main chain is included.

```
("READ" means a bit-shift or rotate to get integer values)
(FUNCTION IGET(hi,lo) reads # bits required for range, adds lo)
START (at leftmost bit of word)
READ 2 bits as $C_t$ (total carbons)
If $C_t$=0 then READ 4 bits as $C_t$ else $C_t$=$C_t$+18 ($C_{19}$-$C_{21}$)
If $C_t$=0 then READ 2 bits as $C_t$ else if $C_t$<16 then $C_t$=$C_t$+3 ($C_4$-$C_{15}$)
If $C_t$<4 then STOP ($CH_4$-$C_3H_8$, 0=error)
READ 4 bits as $C_s$ (0-15 side-chain carbons)
L=$C_t$-$C_s$ (main chain length)
If $C_s$=0 then STOP (n-alkane)
sym=true (initial empty chain symmetric)
lohalf=INT((L+1)/2) (maxloc for 1 most cplx side chain)
If $C_s$>2 then READ next bit as Y else Y=0 (Me/Et only flag=0)
If Y=0 then maxsml=2 else maxsml=MIN($C_s$,4) (max small side-chain size)
cplx=(Y=1 & $C_s$>4 & $C_t$>13) (T or F: 1-2 possible complex side chains)
(Find complex side chains; see NOTE below)
Do while cplx & $C_s$>4
  READ next bit as cplx (1=true)
  If cplx then (get size, length, branches, locant on main)
    $C_t$'=IGET($C_s$,5); lmax=MIN(lohalf-1,$C_s$)
    $C_s$'min=$C_t$'-lmax; $C_s$'max=$C_t$'-3; $C_s$'=IGET($C_s$'max,$C_s$'min)
    L'=$C_t$'-$C_s$'
    If $C_s$'>2 then READ next bit as Y else Y=0 (Me/Et only flag=0)
    If Y=0 then maxsml'=2 else maxsml'=MIN($C_s$',4) (max small on cplx)
    Call SIMPLE (cplx,$C_s$',L',maxsml') (branches)
    loloc=L'+1; $C_s$=$C_s$-$C_t$' (to get loc on main chain)
    If sym & $C_s$<$C_t$' then hiloc=lohalf else hiloc=L-L' (most cplx?)
    loc=IGET(hiloc,loloc)
    If sym then sym=(MOD(L,2)=1 & loc=lohalf)  (mid of odd-L chain)
  Else (remaining are simple)
End(cplx)
(Find simple side chains on main chain)
cplx=false
Call SIMPLE(cplx,$C_s$,L,maxsml)
STOP
END

SUBROUTINE SIMPLE(cplx,$C_s$,L,maxsml)
(1. Get Butyl, propyl and ethyl side chains: "make change")
LENBU(4)/4,3,3,2/; LENPR(2)/3,2/ (length of n,s,i,tBu; n,iPr)
sym=(not cplx)  (possible on main only)
lohalf=INT((L+1)/2) (maxloc for 1 most cplx side chain)
Do size=maxsml,2,-1 while $C_s$>1
  maxgps=INT($C_s$/size) (max alkyl groups of this size)
  ngps=IGET(maxgps,0) (actual #)
  xtype=-1 (init prev type)
  type=size-1 (init to read 2 bits for Bu, 1 Pr, 0 Et)
  Do i=1,ngps while ngps>0 (find Bu, Pr, Et types, locants)
    type=IGET(type,1)  (elim lead zeroes after 1st time)
    If size=4 then loloc=LENBU(type)+1 (n,s,i or t-Bu)
    Else if size=3 then loloc=LENPR(type)+1 (n or i-Pr)
    Else loloc=3  (ethyl)
    If cplx then minloc=1 else minloc=loloc
    If sym & type ne xtype then COMPUTE sym  (incl prev cplx)
    If ngps=1 & sym then hiloc=lohalf (1 of this size only)
    Else if type=xtype then hiloc=loc (max if same type as last)
    Else hiloc=L+1-loloc (1st of new type)
    loc=IGET(hiloc,minloc)
    xtype=type
  End(Do ngps)
  $C_s$=$C_s$-ngps*size (left for next size)
End(Do size)
(2. Locate methyl groups)
nme=$C_s$; nxtfw=2 (field width); if cplx then loloc=1 else loloc=2
If sym then COMPUTE sym (after Et added above)
If nme=1 & sym then hiloc=lohalf else hiloc=L-1
Do while $C_s$>0 & hiloc>loloc
  If nme<4 or $C_s$=1 then method=1 else method=2
  If method=1 then (indiv Me locants)
    nhere=1; meloc=IGET(hiloc,loloc)
    hiloc=meloc (reduce position uncertainty)
  Else (method=2: position count 0-2)
    nhere=IGET(nxtfw,0)  (nxtfw bits read)
    If nxtfw=1 then nhere=nhere+1; nxtfw=2 (1 or 2 here, 0 prev)
    else if nhere=0 then hiloc=hiloc-1 (none here, next)
    else if nhere=1 then nhere=0; nxtfw=1 (none here, 1 or 2 next)
    else nhere=nhere-1 (1 or 2 here)
    hiloc=hiloc-1
  $C_s$=$C_s$-nhere
End($C_s$>0)
nlome=$C_s$ (2-Me groups; 1-Me if cplx)
RETURN
END

FUNCTION IGET(hi,lo)
If hi>lo then nxtfw=INT(LOG2(hi-lo))+1 else nxtfw=0 (next field width)
If nxtfw>0 then READ nxtfw bits as IGET else IGET=0
IGET=IGET+lo
RETURN
END
```

NOTE: To handle nested complex side chains, each subroutine argument would have to be stacked by level: arg → ARG(lvl). The first case by size is the $C_{23}$ alkane 7-(1-*tert*-pentylpentyl)tridecane.

## REFERENCES AND NOTES

(1) Gordon, M.; Kennedy, J. W. *SIAM J. Appl. Math.* **1975**, *28*, 376–398.
(2) Knop, J. V.; Müller, W. R.; Jeričević, Ž.; Trinajstić, N. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 91–99.
(3) Randić, M. *J. Chem. Inf. Comput. Sci.* **1986**, *26*, 136–148.
(4) Read, R. C. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 135–149.
(5) Davidson, S. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 151–155.
(6) IUPAC. *Nomenclature of Organic Compounds*; Pergamon Press: New York, 1979; pp 10–11.
(7) IUPAC. *Nomenclature of Organic Compounds*; Butterworths: London, 1958; pp 8–9.
(8) Morgan, H. L. *J. Chem. Doc.* **1965**, *5*, 107–113.
(9) Lozac'h, N. *Angew. Chem., Int. Ed. Engl.* **1979**, *18*, 887–899.

# Isocodal and Isospectral Points, Edges, and Pairs in Graphs and How To Cope with Them in Computerized Symmetry Recognition

GERTA RÜCKER and CHRISTOPH RÜCKER*

Institut für Organische Chemie und Biochemie, Universität Freiburg, Albertstrasse 21, D-7800 Freiburg, FRG

Received April 3, 1991

It is demonstrated that in certain graphs isospectral edges and *pairs* exist, in analogy to the well-known isospectral points. A *pair* is any relationship between two vertices (an edge is thus a special kind of a pair), and isospectral pairs are pairs which, when arbitrarily but identically perturbed, always yield isospectral graphs. The significance of isospectral points, edges, and pairs is that computer programs for symmetry perception and for graph isomorphism testing tend to encounter difficulties when processing graphs containing such features; they tend to take isospectrality for equivalence by symmetry. It is shown how in the authors' programs TOPSYM and MATSYM these difficulties are overcome by using the newly developed "class matrix procedure".

## INTRODUCTION, DEFINITIONS, AND EXAMPLES

Recognition of constitutional (topological) symmetry in (molecular) graphs is of major interest for any molecular structure processing task, and ever more powerful computer programs for this purpose are still being developped, see refs 1–3 and references cited therein. For example, Balasubramanian et al. recently used their vertex partitioning program to predict the number and intensity ratios of NMR signals.[4] Since their program does not partition the *pairs* of vertices, they are not able to predict the number of coupling constants. A *pair* is a defined relation between two vertices, either a long-range relation or a one-bond relation (edge). For an illustration of the term *pair* see Figure 1.[5]

We recently developped the program TOPSYM (as an aid in the machine generation of IUPAC names for polycyclic compounds[6]) that partitions both the vertices and the *pairs* of vertices in a graph into equivalence classes.[1] The program relies primarily on different entries in the higher powers of