A data structure for family relations

F. Mavaddat and B. Parhami

Department of Mathematics and Computer Science, Arya-Mehr University of Technology, Tehran, Iran

A data structure is proposed which enables efficient determination of family relations of common interest with the minimum amount of information on each individual. The problem of updating information due to births, deaths, marriages and divorces is considered. Algorithms for determining the immediate relatives of each individual are given and a framework is established for writing procedures to determine other relatives.

(Received June 1977)

Due to the rich structure of pointers and universal familiarity of subject, representation of family relations has always provided the students of data structures with an unfailing source of examples and drills. It has also supplied educators with materials for demonstration of points and clarification of concepts in the theory of linked structures (Knuth, 1968; Elson, 1975; Hoare, 1968; Stone, 1972). Its influence on the subject of data structures has been so strong that some kinship terms, such as 'offspring' and 'sibling', are universally used for naming relationships in trees, a very important class of data structures (Knuth, 1968 page 307).

Unfortunately, most treatments of such data structures are simplistic, in the sense that they only partially 'contain' common family relations. We say that a data structure contains a relation if a non-exhaustive procedure can be devised for finding all items having that relation to a given item. Relations such as 'current wives' and 'ex-sisters-in-law' are difficult to compute without a heavily linked structure. Our aim is to propose a sufficiently rich structure which contains all family relations of common interest. Examples of such relations are those of 'mother,' 'brother', 'spouse', 'daughter', 'grandparent', 'uncle', 'nephew', 'cousin', and 'sister-in-law', with qualifications of age, sex, existence, order and halfness, where applicable.

In addition to its academic value, such a study can be used by private and government organisations for tracing family relations. Such relations, and the associated terminology, are also of interest to social anthropologists and linguists (Wallace and Atkins, 1960; Baateni, 1973). Therefore, a better understanding of the structure needed to represent such relations can be of value in their work. In fact, our original interest in this work was triggered by a linguistic study of kinship terms in Farsi (Baateni, 1973).

Functional specifications

Actual family relations are dynamic in nature. The processes of birth, death, marriage and divorce continually create new relations and alter existing ones. Therefore a structure for representing these relations must be designed in view not only of the ability to respond to queries but also of the execution of updating procedures. For this reason, the total system may be viewed as a kinship data base.

Updating information is required as a result of the following events: Birth, death, marriage and divorce.* These four types of events are represented by directives of the type

'**An** *s* **named** *n* **is born to** *p* **and** *q* **on** *d*.' (1)
'*x* **dies on** *d*.' (2)
'*p* **and** *q* **get married on** *d*.' (3)
'*p* **and** *q* **get divorced on** *d*.' (4)

respectively, where p and q uniquely identify a male and a female, $s \in$ {**male, female**}, n is the child's name, x uniquely identifies an individual, and d is the date of event. In the following pages, we will demonstrate that the above mentioned directives can be handled by efficient algorithms for the proposed structure.

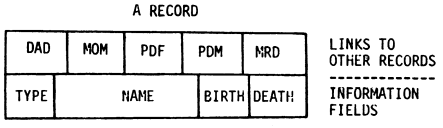
Queries put to the system are of greater variety because of the large number of possible relations. All such queries can, nevertheless, be expressed in one of the following three forms

'**Who are the** *r*'**s of** *p*?' (5)
'**Is** *q* **an** *r* **of** *p*?' (6)
'**What is** *q* **to** *p*?' (7)

where p and q uniquely identify two individuals and r is a member of the set of all possible relations. Queries of type (6) can be handled by answering (5) and searching for q in the resulting set. Queries of type (7) can be handled by searching for positive answers to (6) for different r's of interest. We will, therefore, only consider queries of type (5) in our subsequent discussion.

Since the set of possible values for r in (5) is infinite, it is not possible to prove by exhaustive enumeration that any given structure will enable us to answer all queries. Therefore, the following inductive scheme based on intrinsic properties of family relations will be used.

We define an 'immediate relative' to be a parent, spouse, sibling, or offspring. Any other relative of an individual is obtainable as a 'closer' relative of an immediate relative. Therefore, in the section on retrieval algorithms, we need only give procedures for determining the immediate relatives of an individual. Other relatives can be traced by composite procedures, as will be demonstrated through a number of examples. Special routines can be built into the system for answering



DAD—Link to father's record
MOM—Link to mother's record
PDF—Link to previous descendant of father
PDM—Link to previous descendant of mother
MRD—Link to most recent descendant
TYPE—Record type (**male, female,** or **marriage**)
NAME—Name of record's owner (string)
BIRTH—Date of birth (numeric)
DEATH—Date of death (numeric)

Fig. 1 Format of record for each individual

*It is also possible to consider adoption and abandonment of children as events. However, this would make the algorithms considerably more complex.

queries on common relations. Other queries can be handled with the aid of a conventional language for dealing with lists and pointers.

The proposed structure

It is clearly desirable to have a structure where the information on each individual is kept in a fixed format record with the minimum amount of information. Stated in a different way, we are interested in the minimum uniform information that each person in a society has to have to be able to find all relatives of given relationship to him by repeated interviewing of others. We propose that the information on each individual be kept in a record as in **Fig. 1.**

The same format is used to keep a record of marriages (TYPE field containing **marriage**). A marriage record has DAD and MOM pointing to the records of the married couple and is linked into the list of descendants of both individuals through PDF and PDM pointers, exactly as an offspring's record would. A marriage record can, therefore, be viewed as representing a virtual offspring. The marriage and divorce dates are stored in BIRTH and DEATH fields, respectively.*

In the above structure, the list of offsprings and marriages (list of events) for each individual is maintained and manipulated as a stack, with MRD pointing to its top and PDF or PDM for each stack element pointing down the stack to previous elements depending on whether the stack has a male or female owner. The last element of stack is designated by the null pointer Λ.

Fig. 2 has been constructed to demonstrate the concepts discussed so far. It shows the records for eight individuals and four marriages, with links corresponding to the events on dates D_1 through D_9, also given in Fig. 2.

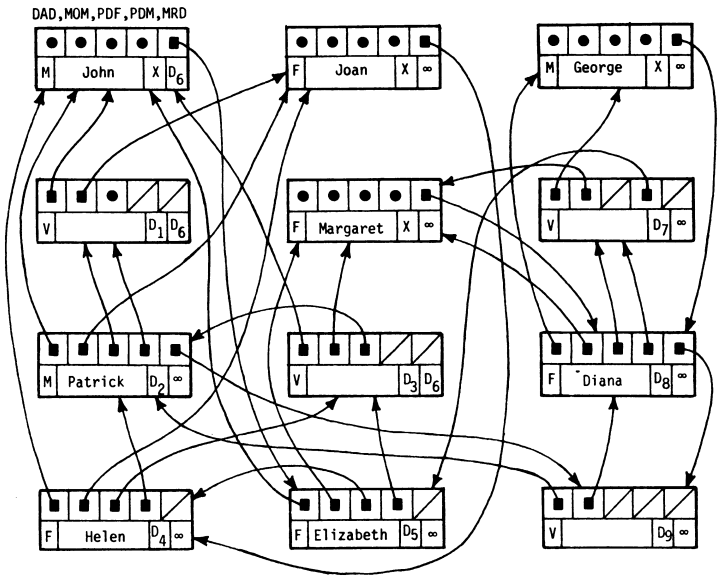
The structure defined so far is not changed except by updating procedures defined in the next section. For the purpose of retrieval operations, different structures are created and manipulated. These are linear lists of nodes with the format shown in **Fig. 3.** Each such list is uniquely identified by a link variable (such as L) pointing to its first node.†

In the algorithms which follow, an individual is always identified by a pointer to his (her) record. The translation from identifying characteristics to location is straightforward if one keeps a list of all individuals, with appropriate structure to facilitate searching. In general, a number of permanent lists may exist for different purposes; e.g. list of marriage records, list of living individuals, etc. If such lists are used, the updating algorithms must be augmented to maintain such lists in a proper manner.

It is worth noting that the proposed data structure may be viewed as a variant of the CODASYL set construct (Date, 1975; CODASYL, 1971) with the difference that sets can self-reference a single file. **Fig. 4,** in which each record is partitioned into two segments called PERSON and LINK, illustrates the set construct much more clearly than Fig. 2. We have deliberately left out MOM and DAD pointers, as well as several information fields, for clarity, since they do not affect the owner/member relationship in a set. The dotted links in Fig. 4 are not present in the actual data structure but their functions are performed by MOM or DAD pointers, depending on the owner's sex. The correspondence between a PERSON and a LINK segment in Fig. 4 is shown informally by associating the person's name with the LINK segment.

*The MRD and NAME fields are unused in a marriage record. They may be used to hold other information regarding the marriage in an actual application.

†We will use upper case letters to designate pointers to lists and lower case letters for pointers to records.



Configuration of Links Immediately after Date D_9.

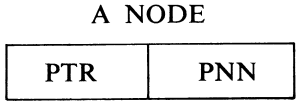
Milestones (in chronological order)

D_1: John marries Joan; his second, her first.
D_2: Patrick is born to John and Joan.
D_3: John marries Margaret; his third, her first (polygamy).
D_4: Helen is born to John and Joan.
D_5: Elizabeth is born to John and Margaret.
D_6: John passes away.
D_7: George marries Margaret; his first, her second.
D_8: Diana is born to George and Margaret.
D_9: Patrick marries Diana.

Legend

☒ Null link
● Unspecified link
X Unspecified information field
V Virtual child

Fig. 2 An example for the proposed data structure



PTR – – Pointer to record
PNN – – Pointer to next node

Fig. 3 A node in a list designating a set of records

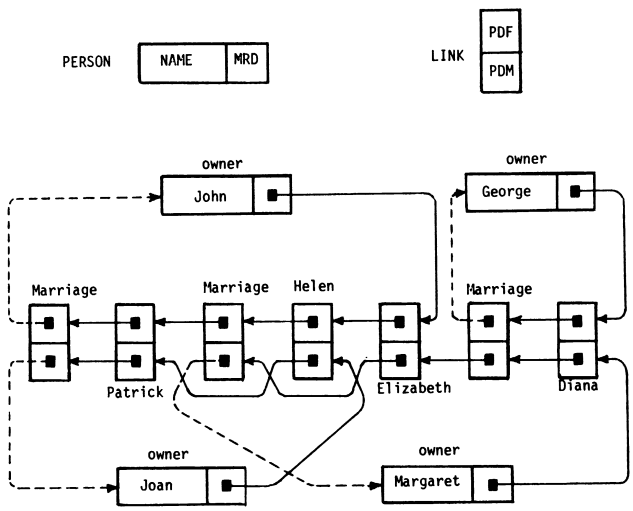


Fig. 4 The proposed data structure as a collection of sets

Updating information

Assuming that the operation $r \leftarrow$ NEWRECORD makes r point to a new (unused) record and updates the pool of free space, we can write the following algorithms for the updating operations. The interpretation of the parameters for these procedures is as in (1), (2), (3), and (4).

birth (s,n,p,q,d) (8)

$r \leftarrow$ NEWRECORD
TYPE $[r] \leftarrow s$; NAME $[r] \leftarrow n$; BIRTH $[r] \leftarrow d$;
DEATH $[r] \leftarrow \infty$
DAD $[r] \leftarrow p$; MOM $[r] \leftarrow q$; MRD $[r] \leftarrow \Lambda$
PDF $[r] \leftarrow$ MRD $[p]$; PDM $[r] \leftarrow$ MRD $[q]$;
MRD $[p] \leftarrow$ MRD $[q] \leftarrow r$

death (x,d) (9)

DEATH $[x] \leftarrow d$; $q \leftarrow$ MRD $[x]$; $s \leftarrow$ TYPE $[x]$
while $q \neq \Lambda$ **do begin**
if TYPE $[q]$ = **marriage and** DEATH $[q] = \infty$
then DEATH $[q] \leftarrow d$
if s = **male then** $q \leftarrow$ PDF $[q]$ **else** $q \leftarrow$ PDM $[q]$
end

marriage (p,q,d) (10)

$r \leftarrow$ NEWRECORD
TYPE $[r] \leftarrow$ **marriage**; BIRTH $[r] \leftarrow d$;
DEATH $[r] \leftarrow \infty$
DAD $[r] \leftarrow p$; MOM $[r] \leftarrow q$
PDF $[r] \leftarrow$ MRD $[p]$; PDM $[r] \leftarrow$ MRD $[q]$;
MRD $[p] \leftarrow$ MRD $[q] \leftarrow r$

divorce (p,q,d) (11)

$r \leftarrow$ MRD $[p]$
while TYPE $[r] \neq$ **marriage or** MOM $[r] \neq q$
set $r \leftarrow$ PDF $[r]$
DEATH $[r] \leftarrow d$

In the above algorithms, it is assumed that all needed records exist within the system. If it is possible, for example, to receive a divorce report without having received the corresponding marriage report previously, our procedures must be modified to take care of such singularities.

Retrieval of information

For the sake of generality, we use upper case identifiers to define retrieval functions on sets* (represented by linear lists of nodes) so that function compositions of the form BROTHERS (parents(p)) can easily be performed. Corresponding to each upper case function (e.g. FUNC) which operates on sets, is a lower case function (e.g. func) which has a record pointer as argument and returns a set as its result. Each set is identified by the name of its corresponding list pointer, and set operations such as UNION(P,Q), INTERSECTION-(P,Q), and DIFFERENCE(P,Q) have their usual meanings.

Any upper case function can be defined recursively in terms of the corresponding lower case function as follows

$Q \leftarrow FUNC(P)$ (12)

if $P = \Lambda$ **then** $Q \leftarrow \Lambda$
else $Q \leftarrow$ UNION (func(PTR[P]), FUNC(PNN[P]))

where Q specifies the result of the function FUNC applied to the set P.† In our subsequent discussion, we will only define lower case functions since the corresponding upper case functions can easily be defined as in (12).

*Field names, which are also upper case identifiers, are distinguished from function names by enclosing their parameters in square brackets.

†A letter 'S' is added to the name of some upper case functions for the sake of readability; e.g. 'FATHERS' corresponds to 'father'.

For convenience, we will be using the operation $Q \leftarrow$ makelist(p) for making a one-element list with p as its sole member. The procedures for finding immediate relatives follow. Several auxiliary functions have also been defined to simplify the algorithms. Sex-symmetric definitions have been omitted for brevity.

$Q \leftarrow$ *father*(p) (13)

$Q \leftarrow$ makelist(DAD[p])

$Q \leftarrow$ *parents*(p) (14)

$Q \leftarrow$ UNION(father(p), mother(p))

$Q \leftarrow$ *male*(p) (15)

if TYPE $[p]$ = **male then** $Q \leftarrow$ makelist(p) **else** $Q \leftarrow \Lambda$

$Q \leftarrow$ *marriage*(p) (16)

if TYPE $[p]$ = **marriage then** $Q \leftarrow$ makelist(p) **else** $Q \leftarrow \Lambda$

$Q \leftarrow$ *flist*(p) (i.e. father's previous events) (17)

$Q \leftarrow$ makelist(p); $r \leftarrow$ PDF $[p]$
if $r \neq \Lambda$ **then** $Q \leftarrow$ UNION(Q, flist(r))

$Q \leftarrow$ *eventlist*(p) (i.e. list of offspring and marriages) (18)

$r \leftarrow$ MRD $[p]$
if $r = \Lambda$ **then** $Q \leftarrow \Lambda$
else if TYPE $[p]$ = **male then** $Q \leftarrow$ flist(r)
else $Q \leftarrow$ mlist(r)

$Q \leftarrow$ *sons*(p) (19)

$Q \leftarrow$ MALES(eventlist(p))

$Q \leftarrow$ *offsprings*(p) (20)

$Q \leftarrow$ UNION(sons(p), daughters(p))

$Q \leftarrow$ *currentmarriage*(p) (21)

if TYPE $[P]$ = **marriage and** DEATH $[p] = \infty$
then $Q \leftarrow$ makelist(p)
else $Q \leftarrow \Lambda$

$Q \leftarrow$ *wives*(p) (22)

if TYPE $[p]$ = **male then**
$Q \leftarrow$ MOTHERS- (MARRIAGES(eventlist(p)))
else $Q \leftarrow \Lambda$

$Q \leftarrow$ *exwives*(p) (23)

if TYPE $[p]$ = **male then** $Q \leftarrow$ MOTHERS-
(EXMARRIAGES(eventlist(p)))
else $Q \leftarrow \Lambda$

$Q \leftarrow$ *spouses*(p) (24)

if TYPE $[p]$ = **male then** $Q \leftarrow$ wives(p) **else** $Q \leftarrow$
husbands(p)

$Q \leftarrow$ *fsiblings*(p) (i.e. siblings from same father) (25)

$Q \leftarrow$ DIFFERENCE(OFFSPRINGS(father(p)),
makelist(p))

$Q \leftarrow$ *siblings*(p) (26)

$Q \leftarrow$ UNION(fsiblings(p), msiblings(p))

$Q \leftarrow$ *fullsiblings*(p) (27)

$Q \leftarrow$ INTERSECTION(fsiblings(p), msiblings(p))

$Q \leftarrow$ *halfsiblings*(p) (28)

$Q \leftarrow$ DIFFERENCE(siblings(p), fullsiblings(p))

$Q \leftarrow$ *brothers*(p) (29)

$Q \leftarrow$ MALES(siblings(p))

$Q \leftarrow$ *fullbrothers*(p) (30)
$Q \leftarrow$ MALES(fullsiblings(p))

Using the conventions outlined so far and with the aid of the defined functions, it is an easy matter to find other relatives through function compositions. Several examples follow.

$Q \leftarrow$ *grandfathers*(p) (31)
$Q \leftarrow$ FATHERS(parents(p))

$Q \leftarrow$ *grandchildren*(p) (32)
$Q \leftarrow$ OFFSPRINGS(offsprings(p))

$Q \leftarrow$ *uncles*(p) (33)
$Q \leftarrow$ BROTHERS(parents(p))

$Q \leftarrow$ *sistersinlaw*(p) (34)
$Q \leftarrow$ UNION(SISTERS(spouses(p)),
WIVES(brothers(p)))

$Q \leftarrow$ *cousins*(p) (35)
$Q \leftarrow$ OFFSPRINGS(SIBLINGS(parents(p)))

$Q \leftarrow$ *nephews*(p) (36)
$R \leftarrow$ UNION(sistersinlaw(p), brothersinlaw(p))
$Q \leftarrow$ SONS(UNION(siblings(p), R))

$Q \leftarrow$ *stepmothers*(p) (37)
$Q \leftarrow$ DIFFERENCE(CURRENTWIVES(father(p)),
mother(p))

Conclusion

In this paper, we have proposed a data structure, with a minimum amount of uniform information on each individual, which is sufficiently rich to enable the determination of all family relations of common interest.

The proposed structure has adequate generality to be applicable to a wide variety of societies with different marriage laws (monogamy, polygamy, and polyandry), religious beliefs, and degrees of permissiveness (e.g. handling children of unmarried parents). The key to this power is the sex-symmetry of records and the introduction of marriage records as virtual offsprings.

Since the answers to queries are presented as sets of individuals, it is possible to apply other qualifying adjectives such as YOUNGEST and LIVING, as functions on the resulting sets to identify those individuals having the desired properties.

In some languages, a different set of terms for family relations exists. For example, Farsi (Persian) has two terms for 'uncle' (father's or mother's brother), two terms for 'aunt', and eight terms for 'cousin'. It is possible to write procedures for determining all such relatives with the proposed structure.

In any real application, the volume of data will be extremely large. Therefore, it is important to consider storage schemes for handling such large volumes of data. One approach would be to take advantage of the clustering property of family relations which occurs for geographical, ethnic, religious, and racial reasons. Thus one expects the structure of Fig. 2 to grow as a collection of loosely linked clusters. Dividing the set of records along the cluster boundaries, one obtains a collection of segments which defines the locality of most searches, i.e. a search which starts in one segment is likely to proceed and end in the same segment. Hence, at any given time, one deals only with a small fraction of the total data base.

A possible extension of the work presented here is in the area of formal definition of kinship terms which may be of interest to semanticists. Also, we have not dealt with the question of efficiency in the retrieval algorithms. Further work may proceed on modifications to the proposed structure or algorithms which would result in improved efficiency.

Acknowledgement

We acknowledge a very fruitful discussion with our colleague Dr. M. Toosi and several helpful comments from anonymous referees.

References

BAATENI, M. R. (1973). Kinship Terms in Persian, *Anthropological Linguistics*, Vol. 15, No. 7, pp. 324-327.
CODASYL. (1971). *CODASYL Data Base Task Group Report*, April 1971.
DATE, C. J. (1975). *An Introduction to Database Systems*, Addison-Wesley, Reading, Mass., pp. 231-243.
ELSON, M. (1975). *Data Structures*, Science Research Associates, Chicago.
HOARE, C. A. R. (1968). Record Handling, in *Programming Languages*, edited by F. Genuys, Academic Press, pp. 291-347.
KNUTH, D. E. (1968). *The Art of Computer Programming*—Vol. 1: Fundamental Algorithms, Addison-Wesley, New York.
STONE, H. S. (1972). *Introduction to Computer Organization and Data Structures*, McGraw-Hill, New York, pp. 211-213.
WALLACE, A. F. and ATKINS, J. (1960). The Meaning of Kinship Terms, *American Anthropologist*, Vol. 62, No. 1.