Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 60 (2007) 596–624
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Towards an intelligent database system founded on the SP theory of computing and cognition

J. Gerard Wolff *

CognitionResearch.org.uk, 18 Penlon, Menai Bridge, Anglesey LL59 5LR, UK

Received 15 March 2006; accepted 22 April 2006
Available online 16 May 2006

Abstract

The SP theory of computing and cognition, described in previous publications, is an attractive model for intelligent databases because it provides a simple but versatile format for different kinds of knowledge, it has capabilities in artificial intelligence, it can function effectively in the face of errors in its input data, and it can function like established database models when that is required.

This paper first describes the SP theory in outline and the computer models in which it is expressed. The main sections of the paper describe, with examples from the SP62 computer model, how the SP framework can emulate other abstract models used in database applications: the relational model (including retrieval of information in the manner of query-by-example, creating a join between two or more tables, and aggregation), object-oriented models (including class-inclusion hierarchies, part-whole hierarchies and their integration, inheritance of attributes, cross-classification and multiple inheritance), and hierarchical and network models (including discrimination networks). Comparisons are made between the SP model and those other models.

The artificial intelligence capabilities of the SP model are briefly reviewed: representation and integration of diverse kinds of knowledge in one versatile format; fuzzy pattern recognition and recognition at multiple levels of abstraction; best-match and semantic forms of information retrieval; various kinds of exact reasoning and probabilistic reasoning; analysis and production of natural language; planning; problem solving; and unsupervised learning. Also considered are ways in which current prototypes may be translated into an 'industrial strength' working system.

Keywords: Intelligent database; Information compression; Multiple alignment; Database model; Relational database; Object-oriented database; Hierarchical database; Network database

1. Introduction

The SP theory is a new theory of computing and cognition developed with the aim of integrating and simplifying a range of concepts in computing and cognitive science, with a particular emphasis on concepts in

* Tel.: +44 1248 712962.
E-mail address: jgw@cognitionresearch.org.uk
URL: www.cognitionresearch.org.uk

0169-023X/$ - see front matter
doi:10.1016/j.datak.2006.04.003

artificial intelligence. It is founded on the conjecture that all kinds of information processing, both natural (in brains) and artificial (in computers), may be understood as compression of information by the matching and unification of patterns. The name 'SP' expresses the idea that information compression promotes *Simplicity* in a body of information (by the removal of redundancy) whilst retaining as much as possible of its non-redundant descriptive *Power*.

The same two concepts lie at the heart of good science—where theories should, as far as possible, combine simplicity with descriptive or explanatory power—and they are a touchstone of success in engineering where, normally, we aim to create systems that are as simple as possible consistent with what they are designed to do. In the world of databases, these ideas were applied early on when it became apparent that, instead of hard coding each new database from scratch (each with its own user interface and methods for storing and retrieving information), an overall simplification of database systems could be achieved without loss of functionality or 'power' by developing a general-purpose 'database management system' that incorporates the features that are needed in every database (like those just mentioned) and could be loaded with different kinds of data according to need.

In a similar way, the SP project has sought to identify the elements that are shared by different aspects of intelligence (pattern recognition, information retrieval, reasoning, learning and so on) and to embody these shared elements in one general-purpose abstract model or 'system' for the representation and processing of knowledge. An overview of the SP theory is presented in [35]. More detail may be found in [37] and in earlier publications cited in those two sources.

1.1. The SP theory and intelligent databases

Amongst other things, the SP system provides an attractive model for database applications, especially those requiring a measure of human-like 'intelligence'. There is, of course, a wide variety of existing database systems that exhibit varying degrees and kinds of intelligence [10] and it is reasonable to ask what may be gained by creating yet another system in that domain. In brief, the main attractions of the SP model in this connection are that:

- It provides an extraordinarily simple yet versatile format for representing knowledge that facilitates the seamless integration of diverse kinds of knowledge.
- It provides a framework for processing that knowledge that integrates and simplifies a range of artificial intelligence functions including fuzzy pattern recognition and recognition at multiple levels of abstraction, best-match and semantic forms of information retrieval, various kinds of exact reasoning and probabilistic reasoning, analysis and production of natural language, planning, problem solving, and unsupervised learning.
- It can function effectively despite errors of omission, commission and substitution in incoming data.
- And it has the ability to emulate established database models when that is required.

As an aid to thinking and as a means of testing ideas, the abstract form of the SP system has been translated into the relatively concrete form of software simulations running on an ordinary computer, described in outline below (Section 2.3). A programme of further development will be needed to translate these prototypes into an 'industrial strength' working system (Section 7).

1.2. Databases and redundancies in knowledge

The SP system aims to compress any given body of knowledge by removing redundancies as far as possible. This is consonant with the generally-acknowledged principle that a 'well-structured' database should minimise redundancies. It is, for example, a recipe for confusion and inconsistencies if there are two different records in a database for 'John Smith' and the details of his address, telephone number and so on. However, it is clear that there is a need for redundancies in knowledge in the form of backup copies (as a safeguard against the risk of catastrophic loss of data) or mirror copies (to speed up processing when data is distributed across a network).

The resolution of this apparent contradiction is to say that, within each copy of a database, redundancies should, as far as possible, be minimised but that multiple copies may be made for the reasons indicated. For

the same reasons, there is no contradiction in the idea that any SP-style database system would be designed to compress its stored knowledge but that backup copies or mirror copies may be made of that compressed knowledge.

1.3. Aims and presentation

The main aims of this paper are

- To describe the SP system in outline, with sufficient detail to make the rest of the paper intelligible.
- To describe how the SP model can emulate other models used in database applications and to compare the SP model with those other models. This is the main substance of the paper.
- To review briefly the artificial intelligence capabilities of the SP model and its relationship with other artificial intelligence systems.
- To consider briefly how current prototypes may be translated into a working system.

This paper does *not* aim to provide a comprehensive view of the SP theory and applications because this has already been provided in [35,37] and earlier publications. The narrower focus of this paper is on the SP system as an intelligent database system and, in particular, the ways in which it may emulate existing database models.

In the next section, the SP theory is described in outline. After that, Sections 3–5 are concerned with the second aim listed above, Section 6 is concerned with the third aim and Section 7 with the fourth.

2. Outline of the SP theory

The SP theory is an abstract model of information processing in *any* kind of system, either natural or artificial. The system is Turing-equivalent in the sense that it can model the workings of a universal Turing machine but it is built from different foundations and it has much more to say about the nature of 'intelligence' than the universal Turing machine or equivalent models such as Lamda Calculus or Post's Canonical System [32]. The entire theory is based on principles of *minimum length encoding* pioneered by Solomonoff [29] and others (see [20]). It is also consonant with the long-established idea that much of human perception and cognition, and the workings of brains and nervous systems, are governed by principles of economy or parsimony (see, for example, [7,25,8,9,30,15]).

In broad terms, the system receives 'New' information from its environment and transfers it to a repository of 'Old' information. At the same time, it tries to compress the information as much as possible by finding patterns that match each other and merging or 'unifying' patterns that are the same.[1] An important part of this process is the building of 'multiple alignments' as described below. This provides the key to several kinds of intelligence to be reviewed in Section 6. At its most general or abstract level, the whole system is conceived as a system for unsupervised learning by the assimilation and compression of raw data to create well-structured knowledge.

2.1. Representation of knowledge

In the SP system, *all* knowledge is stored as arrays of atomic *symbols* in one or two dimensions called *patterns*. In work to date, the main focus has been on one-dimensional patterns (i.e., sequences of symbols) but it is envisaged that, at some stage, the concepts will be generalized to patterns in two dimensions. For present purposes, we may define patterns and symbols as follows:

- A *pattern* is a sequence of symbols bounded by end-of-pattern characters such as '(' and ')', not shown in the examples in this paper.

[1] The term 'unification' in the SP theory means a simple merging of two or more identical patterns, or parts of patterns, to make one. This meaning is different from but related to the meaning of the term in logic.

- A *symbol* is a sequence of non-space characters bounded by space characters at each end or by an end-of-pattern character at one end and one or more space character at the other.
- Any one symbol can be matched with any other symbol and, for any one pair of symbols, the two symbols are either 'the same' or 'different'. No other result is permitted.
- Symbols have no intrinsic meaning such as 'add' for the symbol '+' in arithmetic or 'multiply' for the symbol '×'. Any meaning attaching to an SP symbol takes the form of one or more other symbols with which it is associated in a given set of patterns.
- Each pattern has an associated integer value representing the frequency of occurrence of that pattern in some domain. These values have a role to play in the formation of multiple alignments and also in the calculation of probabilities associated with multiple alignments, as indicated in Section 6. Unless otherwise stated, we may assume that each pattern has the default frequency value of 1.

Although 'flat' sequences of symbols may seem to be a very limited format, the way in which they are processed within the SP system means that they can be used to model a wide range of existing formats for knowledge, including networks and trees of various kinds (including class-inclusion hierarchies, part-whole-hierarchies and discrimination networks), context-free and context-sensitive grammars, condition-action rules, and tables. Some examples will be seen below.

Within the SP system, constructs such as 'variable', 'value', 'type', 'class', 'subclass', 'object', 'iteration', 'true', 'false', and 'negation' are not provided explicitly. However, the effect of these constructs can be achieved by the use of patterns and symbols, and we shall see some examples below.

2.2. Multiple alignments

When one or more New patterns are received, the system tries to find the best possible match between the New pattern(s) and one or more of the Old patterns. The result of this process is the creation of one or more *multiple alignments* as illustrated in Section 2.2.2.

In this research, the concept of multiple alignment has been borrowed from bioinformatics where it means an arrangement of two or more (symbolic representations of) sequences of DNA bases or amino acid residues in rows or columns, with judicious stretching of sequences to bring matching symbols into alignment whilst preserving the order of the symbols. The aim of the computation is to find one or more alignments of a given set of sequences which are, in some sense, 'good'. In broad terms, a 'good' alignment is one with a relatively large number of matches between symbols but alignments may also be evaluated in terms of information theory (see, for example, [5] and Section 2.2.3).

In this area of research, it is widely recognized that the number of possible alignments of symbols is normally too large to be searched exhaustively and that, to achieve a search which has acceptable speed and acceptable scaling properties, 'heuristic' techniques must normally be used, sometimes also called 'local search'. Examples include 'hill climbing' (also called 'descent'), 'beam search', 'genetic algorithms', 'simulated annealing', 'dynamic programming' and others. With these techniques, searching is done in a succession of stages, with a narrowing of the search at each stage using some kind of measure of goodness of alignments to determine where searching should be concentrated. Ideal solutions cannot normally be guaranteed but acceptably good approximate solutions can normally be found without excessive computational demands.

2.2.1. Multiple alignments in the SP system

In the SP framework, the concept of multiple alignment has been modified as follows:

- One or more of the patterns are designated 'New' and the rest are 'Old'.
- A 'good' alignment is one that allows the New pattern or patterns to be encoded economically in terms of the Old patterns. More specifically, alignments are evaluated in terms of principles of *minimum length encoding*, mentioned above. The method of calculation uses the frequency values of patterns mentioned in Section 2.1. Details of the method may be found in [34] and [37, Chapter 3].
- Any one pattern may appear two or more times within one alignment. Notice that two or more *appearances* of one pattern in one alignment is not the same as two or more *copies* of one pattern in the alignment. In the

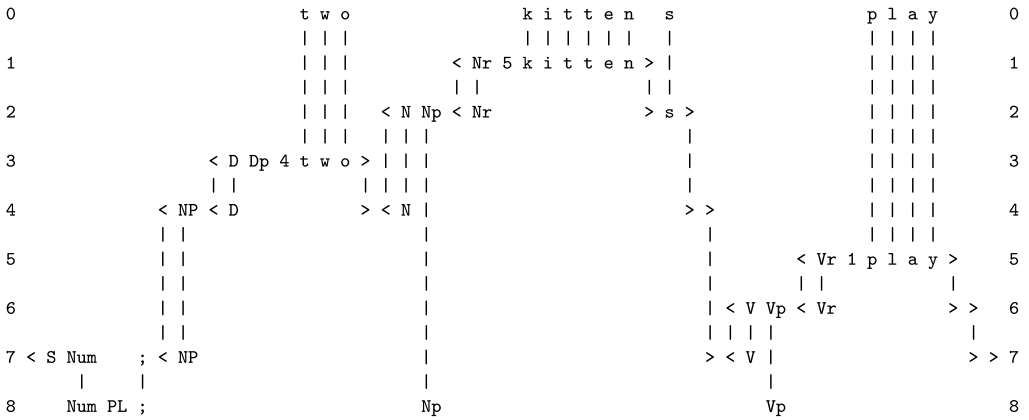


Fig. 1. The best multiple alignment created by SP62 with the sentence 't w o k i t t e n s p l a y' as the New pattern and a set of Old patterns representing grammatical rules.

former case there is just *one* pattern that appears in two or more different parts of a multiple alignment and this means that there are certain constraints on matching that do not apply in the case of two or more copies of one pattern, as explained in [35] and [37, Chapter 3].

2.2.2. Example

A simple example of an SP multiple alignment is shown in Fig. 1. This is the best multiple alignment created by the SP62 computer model (see Section 2.3) with one New pattern ('t w o k i t t e n s p l a y') representing a sentence to be parsed and a set of Old patterns representing grammatical rules.

By convention, the New pattern(s) are always shown in row 0 and a subset of the Old patterns is shown in the remaining rows, one pattern per row. The order of the Old patterns across the rows is arbitrary and without special significance.

This multiple alignment achieves the effect of parsing the sentence into its parts and sub-parts, with each part labelled with its grammatical category. This can be seen most clearly if we reduce each column in the multiple alignment to a single symbol and thus collapse all the rows into a single sequence like this

```
< S Num PL ; < NP < D Dp 4 t w o > < N Np < Nr 5 k i t t e n > s >>
  < V Vp < Vr 1 p l a y >>>
```

From this sequence we can see that 't w o' has been classified as a 'determiner' ('D'), 'k i t t e n s' has been classified as a 'noun' ('N') (divided into a root ('Nr') and a plural suffix ('s')) and the two words together form a 'noun phrase' ('NP'). The word 'p l a y' has been identified as a 'verb' ('V'), and the noun phrase followed by the verb form a sentence ('S'). Incidentally, symbols like '4', '5' and '1' in this example are needed for the calculation of a 'compression score' for each alignment, as outlined in Section 2.2.3.

The symbols 'Num PL ;' mark the sentence as a 'plural' sentence ('PL') which means that the 'plural' noun 'k i t t e n s' (marked with the symbol 'Np') must be followed by the plural verb 'p l a y' (marked with the symbol 'Vp'). The pattern 'Num PL ; Np Vp', shown in row 8 of Fig. 1 expresses this dependency between the plural subject and the plural verb.[2]

The multiple alignment in Fig. 2 shows how the system can parse the sentence successfully despite errors in the sentence, including an error of omission (the missing 'w' in 't w o'), an error of commission (the added 'x' in 'p l a y') and an error of substitution (the replacement of 'n' in 'k i t t e n s' with 'm'). Another example of how the system can cope with errors in the data supplied to the system is presented in Section 5.1.

[2] With sentences that contain subordinate clauses, the main dependencies can be distinguished from the subordinate dependencies by the use of additional symbols to mark each context.

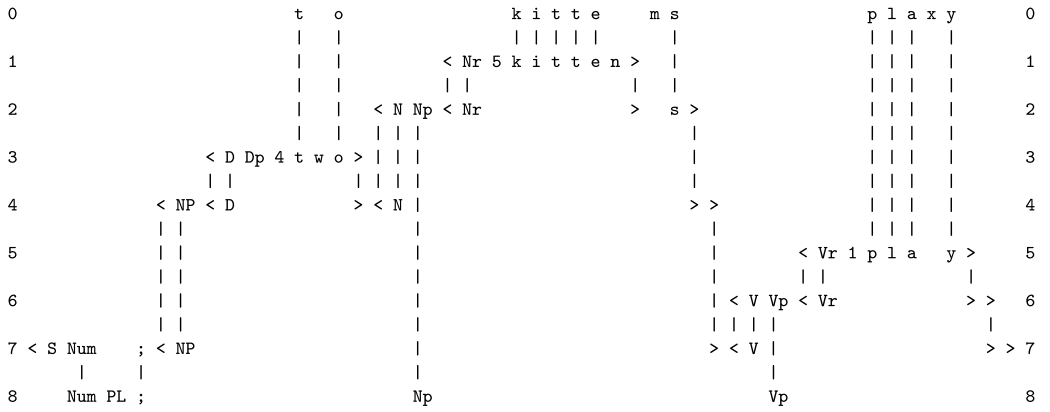


Fig. 2. The best multiple alignment created by SP62 with the same set of Old patterns and the corrupted sentence 't o k i t t e m s p l a x y'.

More complex and more refined examples of natural language processing with the SP system may be found in [34] and [37, Chapter 5].

2.2.3. Evaluation of multiple alignments

In the SP system, each multiple alignment is given a 'compression score' or 'compression difference' calculated as

$$CD = B_N - B_E \tag{1}$$

where B_N is the total number of bits in those symbols in the New pattern that form hits with Old symbols in the alignment and B_E is the total number of bits in the code pattern ('encoding') that has been derived from the alignment as described in [35] and [37, Chapter 3]. In general, the system tries to create multiple alignments with CD values that are as high as possible.

2.3. Computer models

In the development of the SP theory, computer models have been created as a way to reduce vagueness and inconsistencies in the theory, as a way to verify that the system really does work according to expectations, and as a means of demonstrating what the system can do. Two main models have been developed to date:

- SP62 is a partial model of the system that builds multiple alignments from New and Old patterns. It is an enhanced version of SP61 described in [34].[3] This model does not attempt any learning and it does not add any patterns to its repository of Old patterns. All the Old patterns in the model must be supplied by the user when the program starts. This model is relatively stable and provides all the examples in this paper.
- SP70 is an augmented version of SP61 that builds multiple alignments and can learn by adding system-generated patterns to its repository of Old patterns (see [36] and [37, Chapter 9]). This model has already demonstrated significant capabilities for unsupervised learning—creating plausible grammars from sets of sentences—but further work is needed to realize the full potential of the model.[4] Development of the model will aim to produce a system with a robust capability for creating well-structured knowledge from raw data, semi-structured data or badly structured data. Mature versions of the system should also be able to abstract significant patterns or 'rules' from knowledge in the manner of data mining applications.

[3] The main difference between SP62 and SP61 is that the former can form alignments with one *or more* New patterns in one alignment whereas SP61 is restricted to just one New pattern in each alignment.

[4] There is a significant body of evidence, beyond the scope of this paper, that shows a close connection between information compression and the creation of knowledge structures that conform to human intuitions about what is the 'correct' or 'natural' structure of the raw data. ZIP programs like WinZip or PKZIP are designed for speed of execution rather than maximising compression, and for that reason they are less successful at picking out 'natural' structures than learning programs that compress information more thoroughly.

At the heart of the SP models is a version of dynamic programming (see [28]) that allows the system to find 'good' full and partial matches between patterns. The technique that has been developed in the SP models has advantages compared with standard techniques for dynamic programming: it can process arbitrarily long patterns without excessive demands on memory, it can find many alternative matches, and the 'depth' or thoroughness of searching can be determined by the user [31].

Multiple alignments are built recursively by pairwise alignment of basic patterns and previously-created alignments, starting with the New and Old patterns supplied at the outset. At each stage, the best alignments are selected for further processing and the remainder are discarded.

More detail may be found in [35], in [37, Chapter 3], and in earlier publications. Source code for the SP62 and SP70 models may be downloaded from www.cognitionresearch.org.uk/sp.htm#SOURCE-CODE.

3. The relational model

This section and the two that follow describe how the SP model may achieve the effect of popular database models used in 'mainstream' data processing applications. This section discusses the relational model, Section 4 is concerned with object-oriented models and Section 5 considers the hierarchical and network models. The discussion in each of these three sections is not exhaustive: a comprehensive discussion of each area would probably require a whole paper to itself.

Consider a typical table from a relational database like the one shown in Table 1 (from the *DreamHome* example in [18, p. 80]). The same information can be represented using SP patterns, as shown in Fig. 3.

In Fig. 3, the symbols '0' in the first pattern and '1' in the second pattern have a role in the scoring of multiple alignments like the symbols '4', '5' and '1' mentioned in Section 2.2.2.

Readers who are familiar with XML [13] will see that the way in which tables are represented with SP patterns is essentially the same as how they are represented in XML. Each pattern begins with a symbol '<staff>' that identifies it as a pattern representing a member of staff and there is a corresponding symbol, '</staff>', at the end. Likewise, each field within each pattern is marked by start and end symbols such as '<first_name> ... </first_name>' and '<last_name> ... </last_name>'.

Although SP patterns may take on the style of XML elements, other styles may also be used. For example, '<staff> ... </staff>', '<first_name> ... </first_name>' and '<last_name> ... </last_name>' may be replaced by symbols such as 'staff ... #staff', 'first_name ... #first_name' and 'last_name ... #last_name'. As can be seen from the multiple alignments shown in Fig. 1, the beginnings and ends of patterns may also be marked with angle brackets ('<' and '>'). Another point in this connection is that start and end tags are not an obligatory feature of SP patterns: in many applications, some or all of the patterns may not have those kinds of markers at all.

At first sight, the SP (and XML) representation of a table is much more long-winded and cumbersome than the representation shown in Fig. 3. But a table in a relational database—as it appears on a computer screen or a computer print-out—is a simplified representation of what is stored in computer memory or on a computer disk. In relational database systems, the 'internal' representation of each table contains memory pointers or tags that are close analogues of symbols like '<staff>', '</staff>', '<first_name>' and '</first_name>' that appear in the SP and XML representations. In short, the SP representation is essentially the same as the 'internal'

Table 1
A typical table in a relational database representing members of staff in the fictitious *DreamHome* property management company (reproduced from [18, Fig. 3.3, p. 80] with permission from Pearson Education Limited)

Staff no.	First name	Last name	Position	Sex	DoB	Salary	Branch no.
SL21	John	White	Manager	M	1-Oct-45	30,000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12,000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18,000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24,000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

```
<staff> 0
     <staff_no> SL21 </staff_no>
     <first_name> John </first_name>
     <last_name> White </last_name>
     <position> Manager </position>
     <sex> M </sex>
     <dob> 1-Oct-45 </dob>
     <salary> 30000 </salary>
     <branch_no> B005 </branch_no>
</staff>

<staff> 1
     <staff_no> SG37 </staff_no>
     <first_name> Ann </first_name>
     <last_name> Beech </last_name>
     <position> Assistant </position>
     <sex> F </sex>
     <dob> 10-Nov-60 </dob>
     <salary> 12000 </salary>
     <branch_no> B003 </branch_no>
</staff>
```

Fig. 3. Two SP patterns representing the first two rows of the table shown in Table 1.

representation of a table in a relational database. The way that tables are printed out or displayed on a computer screen is largely a cosmetic matter and there is no reason why tables in an SP database should not be printed or displayed in the conventional style.

3.1. Retrieval of information: query by example

In the SP system, the most natural way to retrieve information is in the manner of 'query-by-example'. To achieve this, a set of patterns like those shown in Fig. 3 are stored as Old patterns and the query is created as a New pattern in the same format as the stored patterns but with fewer symbols. For example, if we wish to identify all the female staff at branch number B003, our query pattern would be '<staff> <sex> F </sex> <branch_no> B003 </branch_no> </staff>' or even '<staff> F B003 </staff>'.[5]

Given '<staff> <sex> F </sex> <branch_no> B003 </branch_no> </staff>' as the New pattern and patterns like those in Fig. 3 as Old patterns, SP62 creates a variety of multiple alignments but only two of them match all the symbols in the New pattern. These two multiple alignments—shown in Fig. 4—identify all the female staff in branch B003 ('Susan Brand' and 'Ann Beech'), as required.[6]

Of course, there is no need for the results of the user's query to be displayed in the manner shown in Fig. 4. As with the representation of tables, there is no reason in principle why information should not be displayed or printed in whatever format is convenient. In general, the building of multiple alignments is a computational technique, not a means of presenting information to the user. Multiple alignments would normally be hidden from view, although there may be occasions when users may wish to see them.

3.1.1. Retrieving information from two or more tables

With relational databases, it is of course quite usual for a single query to retrieve information from two or more tables. This subsection shows how this can be done in the SP model with an example corresponding to a simple join between two tables.

[5] Notice that 'F' cannot form a match with 'Ford' (in Table 1) because of the rule that symbols are matched in an all-or-nothing manner. If we wish to achieve fuzzy matching between words such as 'computer' and 'commuter' then each such word must be represented so that each of its constituent letters are symbols, e.g., 'c o m p u t e r' and 'c o m m u t e r'.

[6] Compared with Fig. 1, these two alignments are each rotated through 90°. In this case, the New pattern(s) are always shown in column 0 and Old patterns are shown in the remaining columns, one pattern per column. In later examples we shall see multiple alignments with two more Old patterns and in this case the order of the Old patterns across the columns is arbitrary and without significance.

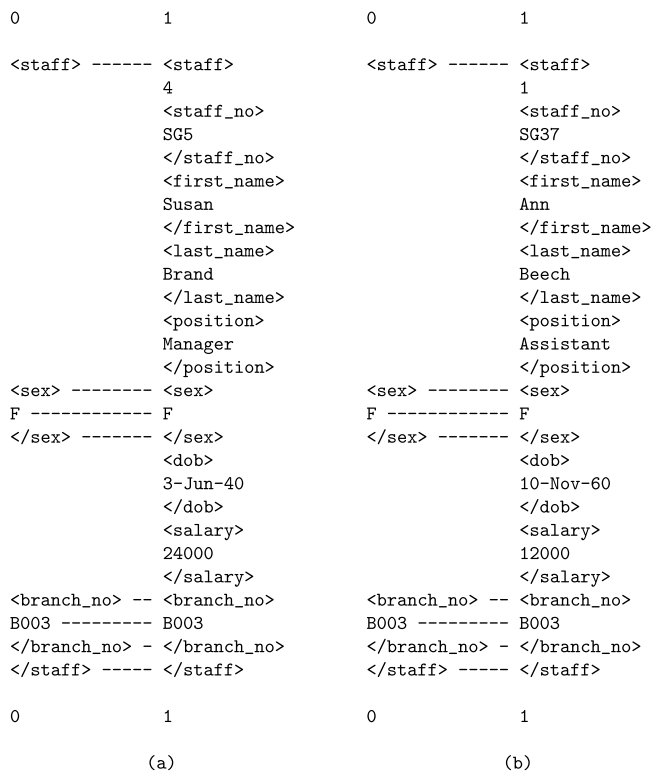


Fig. 4. Two multiple alignments created by SP62 showing how the system can retrieve information from a simulated relational database as described in the text.

In the *DreamHome* example [18, p. 80], there is one table for clients and another for viewings of properties by clients. If we wish to know which clients have viewed one or more properties and the comments they have made (Example 5.24 in [18, p. 138]), we may achieve this with an SQL query like this:

```
SELECT c.client_no, first_name, last_name, property_no, comment
FROM Client c, Viewing v
WHERE c.client_no = v.client_no;
```

In the SP model, an equivalent effect can be achieved by creating multiple alignments like the one shown in Fig. 5. This is one of the five best multiple alignments created by SP62 with the pattern '<viewing> <first_name> </first_name> <last_name> </last_name> <property_no> </property_no> <comment> </comment> </viewing>' in New and a set of patterns corresponding to the two tables in Old.

These five best multiple alignments—the only ones that match all the symbols in the New pattern—answer the original query because each one shows details of one viewing (in column 1) with details of the client who is doing that viewing (in column 2).

If the system is to build multiple alignments like the one shown in Fig. 5, it is necessary for each pattern representing a viewing to refer to the client as '<client> ... </client>' rather than '<client_no> ... </client_no>' (where '...' represents a client number such as 'CR76'). This allows the system to access details of the client such as 'first_name' and 'last_name'.

The key idea in this example is that one pattern (the pattern for a 'viewing' shown in column 1 of Fig. 5) contains a 'reference' to another pattern (the pattern for a 'client' shown in column 2) in the form of the pair of symbols '<client> ... </client>'. It should be clear that this idea can be generalised so that the system can, if required, retrieve information from three or more tables simultaneously (as, for example, in Fig. 8).

```
0                1                2

<viewing> ------ <viewing>
                 11
                 <client> ------- <client>
                                  6
                                  <client_no>
                 CR76 ----------- CR76
                                  </client_no>
<first_name> ------------------- <first_name>
                                  John
</first_name> ----------------- </first_name>
<last_name> ------------------- <last_name>
                                  Kay
</last_name> ------------------ </last_name>
                                  <tel_no>
                                  0207-774-5632
                                  </tel_no>
                                  <pref_type>
                                  Flat
                                  </pref_type>
                                  <max_rent>
                                  425
                                  </max_rent>
                 </client> ------ </client>
<property_no> -- <property_no>
                 PG4
</property_no> - </property_no>
                 <view_date>
                 20-Apr-01
                 </view_date>
<comment> ------ <comment>
                 too
                 remote
</comment> ----- </comment>
</viewing> ----- </viewing>

0                1                2
```

Fig. 5. One of the five best multiple alignments created by SP62 with the pattern shown in column 0 in New and patterns representing tables for clients and viewings in Old.

3.2. Retrieval of information: query languages

As noted above, the most natural way to use the SP system is in the manner of query-by-example. However, users who are familiar with query languages like SQL may wish to stick with what they know.

The SP system does not, in itself, provide SQL or any other query language. However, the system has proved to be effective in the processing of natural languages (see Section 2.2.2, [34] and [37, Chapter 5]) and, since artificial languages like SQL are very much simpler than any natural language, there is reason to believe that SQL and other query languages may be modelled within the SP system. If a query language is deemed necessary, it should be possible to specify the syntax of such a language using SP patterns and to process them within the multiple alignment framework to achieve information retrieval as required. These are matters requiring further investigation.

3.3. Database schemas

A database schema describes the structure of a database, normally at three levels of abstraction:

- At the highest level there may be one or more 'external' schemas or 'subschemas' each one representing a user-oriented 'view' of the data. One user may be able to see staff salaries while another may not have that privilege. One user may see a person's date of birth while another may see their age calculated from their date of birth and the current date.

- At the intermediate level is a 'conceptual' schema that describes the 'entities', 'attributes', 'relationships' and integrity constraints in the database.
- At the lowest level, is an 'internal' schema that describes the data in terms of files, indices and other machine-oriented concepts.

How do these ideas relate to the SP system and how it may be used in database applications?

With regard to the lowest level of abstraction, it is intended that users of the system should be aware of nothing below the level of patterns and symbols. These concepts are more abstract than concepts relating to the underlying hardware but, in themselves, they do not specify any structure of entities, attributes or the like. They may be seen to lie at a level of abstraction between the conceptual schema and the internal schema.

Given the capabilities of an SP system for building multiple alignments, the medium of SP patterns and symbols is a versatile format for describing entities, attributes and relationships. As we shall see in Section 4, concepts of that kind can be described at multiple levels of abstraction. The higher levels of abstraction are similar to abstract descriptions that are provided in the conceptual schema of other database systems.

With regard to integrity constraints such as "no salary shall be less than zero" or the kind of calculation needed to show a person's current age rather than their date of birth, these things can in principle be implemented in the SP system because the system is Turing-equivalent as noted in Section 2. That said, current prototypes are not yet a rival for procedural languages like COBOL or C++ or functional languages like IPL or ML. On short-to-medium timescales, it would probably be more practical to use a conventional programming language (with arithmetic functions) in conjunction with the SP system in much the same way that such languages are often used with relational database systems. Issues of this kind are briefly discussed in Section 7.

3.4. Aggregation

In connection with databases, the term 'aggregation' has two main meanings:

- An 'aggregate function' summarises results from a set of records or performs a calculation on a set of values and returns a single value. For example, with respect to Tables 1–3, an aggregate function may answer questions like 'What is the average rent of properties in each branch?', 'How many properties exist in each city?', or 'What is the number of houses and the number of flats in each city?'.
- The term 'aggregation' is also described by Connolly and Begg [18, p. 371] as follows:

  > A relationship represents an association between two entity types that are conceptually at the same level. Sometimes we want to model a 'has-a' or 'is-part-of' relationship, in which one entity represents a larger entity (the 'whole'), consisting of smaller entities (the 'parts'). This special kind of relationship is called *aggregation* [12].

In itself, the SP system has no concept of number. So it cannot answer the kinds of numerical questions illustrated above. However, in the same way that the system may be supplied with the rules of SQL (Section 3.2), it should be possible at some stage to supply the system with patterns that define the rules of arithmetic and it may then be possible to implement aggregate functions as described above. On shorter timescales, as

Table 2
A table showing branches of the *DreamHome* property management company (reproduced from [18, Fig. 3.3, p. 80] with permission from Pearson Education Limited)

Branch no.	Street	City	Postcode
B005	22 Deer Rd.	London	SW1 4EH
B007	16 Argyll St.	Aberdeen	AB2 3SU
B003	163 Main St.	Glasgow	G11 9QX
B004	32 Manse Rd.	Bristol	BS99 1NZ
B002	56 Clover Dr.	London	NW10 6EU

Table 3
A table showing properties for rent from the *DreamHome* property management company (reproduced from [18, Fig. 3.3, p. 80] with permission from Pearson Education Limited)

Property no.	Street	City	Postcode	Type	Rooms	Rent	Owner no.	Staff no.	Branch no.
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St.	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St.	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd.	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd.	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr.	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

suggested above and in Section 7.4, it is probably more practical to form a hybrid between the SP system and an ordinary programming language with built-in arithmetic functions.

With regard to the second meaning of 'aggregation', the entity-relationship diagram in Fig. 6 contains two examples: the part-whole relationship between members of staff in the *DreamHome* property management company and the branch where they work; and the part-whole relationship between properties available for rent and the branch that manages them.

The way in which aggregation is represented in the relational model can be seen in Tables 1–3. Each branch of the *DreamHome* company has a branch number shown in the first column of Table 2. The one-to-many relation between each branch and the staff that work at that branch is expressed by assigning a branch number to each member of staff, as shown in the last column of Table 1. In a similar way, the one-to-many relation between each branch and the properties that are managed by that branch is implicit in the way a branch number is assigned to each property, as shown in the last column of Table 3.

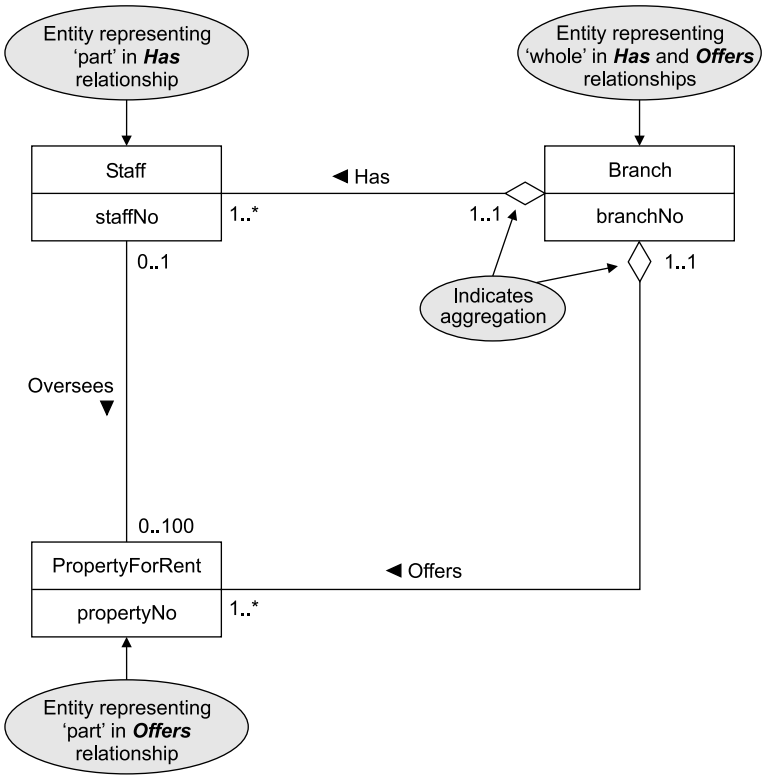


Fig. 6. Examples of aggregation: Branch *Has* Staff and Branch *Offers* PropertyForRent (redrawn from [18, Fig. 12.9, p. 372] with permission from Pearson Education Limited).

The way in which aggregation may be represented in the SP framework is similar. For example, the first row of Table 1 may be represented by pattern (a) in Fig. 7, the first row of Table 2 may be represented by pattern (b) in the same figure and the first row of Table 3 may be represented by pattern (c) in the figure. Notice how the pattern for 'staff' shows the branch where that member of staff is based and how the pattern for 'property' shows the branch that is responsible for that property.

Readers will notice that Tables 2 and 3 are not fully normalised because the fields for 'Street', 'City' and 'Postcode' appear in both these tables and could be abstracted to form a new table representing some such concept as 'Address'. This idea is mirrored in Fig. 7 where pattern (d) represents the address fields for pattern (b)—with the symbol 'addr1' providing a link between these two patterns—and pattern (e) represents the address fields for pattern (c)—with the symbol 'addr6' as the link between them.

A point to notice about pattern (a) in Fig. 7 is that it is slightly different from how the same member of staff is represented in Fig. 3. As with the example of union between two tables (Section 3.1.1), a category name is used instead of a field name when one pattern references another. In this case, the branch where any given member of staff is based is referenced as '<branch> ... </branch>' rather than '<branch_no> ... </branch_no>', and likewise for the last field for each pattern representing a property.

As with the example in Section 3.1.1, the system may create multiple alignments containing patterns corresponding to two or more tables. Let us suppose that we wish to know whether the Glasgow branch of the *DreamHome* company has any properties on its books in the 'G12' area of the city. Given a database of Old

```
(a) <staff> 0
        <staff_no> SL21 </staff_no>
        <first_name> John </first_name>
        <last_name> White </last_name>
        <position> Manager </position>
        <sex> M </sex>
        <dob> 1-Oct-45 </dob>
        <salary> 30000 </salary>
        <branch> B005 </branch>
    </staff>

(b) <branch> 6
        <branch_no> B005 </branch_no>
        <address> addr1 </address>
    </branch>

(c) <property> 11
        <property_no> PA14 </property_no>
        <address> addr6 </address>
        <type> House </type>
        <rooms> 6 </rooms>
        <rent> 650 </rent>
        <owner_no> CO46 </owner_no>
        <staff_no> SA9 </staff_no>
        <branch> B007 </branch>
    </property>

(d) <address> 17
        addr1
        <street> 22 Deer Rd </street>
        <city> London </city>
        <post_code> SW1 4EH </post_code>
    </address>

(e) <address> 22
        addr6
        <street> 16 Holhead </street>
        <city> Aberdeen </city>
        <post_code> AB7 5SU </post_code>
    </address>
```

Fig. 7. Examples of SP patterns representing the information in Tables 1–3. Pattern (a) represents the first row of Table 1, pattern (b) represents the first row of Table 2, and pattern (c) represents the first row of Table 3. Patterns (d) and (e) are examples of patterns representing the address fields of Tables 2 and 3.

patterns like those shown in Fig. 7 we may present our query as a New pattern like this: '<property> G12 <branch> Glasgow </branch> </property>'.

```
0             1              2               3              4

<property> ----------------- <property>
                             16
                             <property_no>
                             PG16
                             </property_no>
               <address> ---- <address>
               27
               addr11 ------- addr11
               <street>
               5
               Novar
               Dr
               </street>
               <city>
               Glasgow
               </city>
               <post_code>
G12 ---------- G12
               9AX
               </post_code>
               </address> --- </address>
                              <type>
                              Flat
                              </type>
                              <rooms>
                              4
                              </rooms>
                              <rent>
                              450
                              </rent>
                              <owner_no>
                              CO93
                              </owner_no>
                              <staff_no>
                              SG14
                              </staff_no>
<branch> ------------------- <branch> ---------------------- <branch>
                                                             8
                                                             <branch_no>
                              B003 -------------------------- B003
                                                             </branch_no>
                                              <address> ---- <address>
                                              19
                                              addr3 -------- addr3
                                              <street>
                                              163
                                              Main
                                              St
                                              </street>
                                              <city>
Glasgow ------------------------------------- Glasgow
                                              </city>
                                              <post_code>
                                              G11
                                              9QX
                                              </post_code>
                                              </address> --- </address>
</branch> ------------------ </branch> ---------------------- </branch>
</property> ---------------- </property>

0             1              2               3              4
```

Fig. 8. The best multiple alignment created by SP62 with the New pattern '<property> G12 <branch> Glasgow </branch> </property>' and a set of Old patterns like those shown in Fig. 7.

In this case, there are only two multiple alignments that match all the New symbols and, together, they say, in effect, that properties 'PG16' and 'PG21' are in the G12 area of the city. The best of those two multiple alignments is shown in Fig. 8.

3.5. Other issues

This section briefly considers two other issues relating to the relational model and how it may be modelled within the SP framework.

3.5.1. Referential integrity

Connolly and Begg [18, p. 458] write that:

> Referential integrity means that if the foreign key contains a value, that value must refer to an existing tuple in the parent relation.

Here, 'foreign key' means a reference number in one table (the 'child' relation) that identifies a tuple in another table (the 'parent' relation). For example, the reference number 'SL41' is a 'foreign key' within the *Staff No.* column of Table 3 (the child relation) and it identifies the member of staff 'Julie Lee' in the last row of Table 1 (the parent relation).

Issues that arise in connection with referential integrity include:

- Is it permissible to show a null value for a foreign key (as in the *Staff No.* field of the third row of Table 3)?
- What rules should be applied when a foreign key is inserted into a table or updated or deleted?
- Are there any 'enterprise constraints' or 'business rules' that govern foreign keys? For example, it might be decided within the *DreamHome* company that no member of staff should be responsible for more than 100 properties at any one time.

How do these concepts and issues relate to the SP framework? Depending on timescales, there are two kinds of answer:

- *On longer timescales*—when the SP system is fully developed—it is anticipated that the system will have robust capabilities for unsupervised learning (as outlined in the introduction to Section 2) so that it will have the ability to receive any kind of input and to organise it automatically to create a body of well-structured knowledge within which redundancies are minimised. It is likely that this mature version of the system will be designed to ensure that, for any symbolic reference from one part of the stored knowledge to another, there would always be at least one pattern containing a copy of that reference as an identifier—a kind of referential integrity that is rather similar to the concept described above. But there is no reason to exclude null values in some contexts, like the empty slot in the third row of the *Staff No.* column of Table 3.
- *On shorter timescales*, partial versions of the system may be developed without any ability to create well-structured knowledge automatically but with an ability to build multiple alignments which would enable it to perform such tasks as fuzzy pattern recognition, best-match retrieval of information, various kinds of reasoning, and so on. Since these versions of the SP system would not be able to learn for themselves, it would be necessary to provide their knowledge ready made and it would be up to the user to ensure that there were no 'dangling pointers' in the knowledge—that for every reference there would be at least one referent. That said, it is entirely feasible to create a hybrid between the core SP system and one or more conventional programming languages (see Section 7.4) and in that case it would be easy enough to provide functions or procedures to check the referential integrity of any given body of knowledge or, perhaps, to enforce the rules for referential integrity (including business rules) within the editor used for adding, modifying or deleting knowledge stored in the SP database.

3.5.2. Normalisation

'Normalisation' of databases is a large topic and it would take us too far afield to discuss it fully in this paper. What follows is a summary of how the SP framework may relate to concepts of normalisation.

An important part of the process of normalising a set of relational database tables is identifying 'functional dependencies' between 'attributes' (the names of columns):

> For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted $A \rightarrow B$), if each value of A is associated with exactly one value of B. (A and B may each consist of one or more attributes). [18, p. 379].

In creating a database in 'third' normal form, for example, one should aim to create tables in which all the non-key columns are functionally dependent on the one or more columns that serve as the key and in which there are no functional dependencies amongst those non-key columns.

Organising the tables of a relational database in this kind of way helps to avoid repetition of information (redundancies) in the database. As an example, consider what would happen if Table 1 (describing members of staff) were to be merged with Table 2 (describing branches of the company) to form a single table. The merged table would contain a row for each member of staff but, in every case where two or more members of staff work at the same branch, the address of the branch would be repeated redundantly for each of those members of staff. The merged table would not conform to the requirements for third normal form because, while the column for *Staff No.* would provide a key for the table (with every other column being functionally dependent on it), there would also be a functional dependency between the non-key column headed *Branch No.* and each of the non-key columns for *Street*, *City* and *Postcode*.

Apart from making the database bigger than it would otherwise be, redundancies in a database can lead to inconsistencies when information in the database is updated. If, for example, Tables 2 and 3 are merged in the way that we just considered and if one branch of the *DreamHome* company were to move to another address, the new address may be entered correctly for some members of staff at that branch but not for others. This kind of inconsistency can be avoided if the table describing members of staff is *not* merged with the table describing branches of the company.

Another example, touched on in Section 3.4, is the way in which the organisation of Tables 2 and 3 may be improved if the columns that appear in both tables (with the headings *Street*, *City* and *Postcode*) are abstracted and placed in a separate table representing addresses. The weakness of the two tables as they stand now may be seen most clearly if, for some reason, the building that houses one of the branches of the *DreamHome* company were to be offered for rent. In that case, the address of that building would be duplicated, once in Table 2 and again in Table 3. As before, there is a risk of inconsistencies appearing when information is replicated like that.

How do these ideas relate to the SP concepts? The obvious connection is that compression of information by the removal of redundancies is bedrock in the SP system and that, when the process of compression is reasonably thorough, it leads to the formation of knowledge structures that conform to our intuitions about what it means for knowledge to be well structured.

The SP70 computer model has already demonstrated a capability for forming well-structured grammars from sets of sentences (see [36] and [37, Chapter 9]). As indicated earlier, a goal of the research is to develop the system to the point where it may receive any kind of input and convert it automatically into a body of knowledge that is properly normalised and correspondingly well structured.

3.6. Comparison between the SP model and the relational model

One of the attractions of the relational model—and perhaps the main reason for its popularity—is the simplicity of the idea of storing all database knowledge in tables. This format is very suitable for much of the knowledge to be stored in typical data processing applications but it is by no means 'universal'. It is not, for example, a good medium for representing any kind of grammar or the kinds of if-then rules used in expert systems. It can be used to represent the kinds of hierarchical structure associated with object-oriented design but it has shortcomings in this connection, as we shall see in the next section.

The three main differences between the SP model and the relational model are these:

- *Simplicity and expressive power.* The SP model provides a format for knowledge that is even simpler than in the relational model. Although this simplification may seem relatively slight, it has a dramatic impact on

what can be represented in the system. Many kinds of knowledge that are outside the scope of the relational model can be accommodated in the SP system and, as we shall see, it overcomes weaknesses of the relational model in representing hierarchical structures. At the same time, it can accommodate the relational model when that is required.

- *Versatility.* The relational model is designed purely for the storage and retrieval of knowledge. By contrast, the SP model can, in addition, support a range of different kinds of intelligence, to be reviewed briefly in Section 6.
- *Robustness in the face of errors*. The form of 'dynamic programming' that is at the heart of the SP models (mentioned in Section 2.3) gives them a robust ability to find 'good' multiple alignments despite errors of omission, commission or substitution. By contrast, relational database management systems can be rather inflexible in their systems for matching patterns, and they can be relatively lacking in the ability to fail gracefully when details are not exactly right.

4. Object-oriented concepts

Since the invention of the Simula language for programming and simulation in the 1960s [11], there has been a growing recognition of the value of organising software and databases into hierarchies of 'classes' and 'subclasses', with 'inheritance' of 'attributes' down each hierarchy to individual 'objects' at the bottom level. An associated idea is that any object may be structured into a hierarchy of 'parts' and 'subparts' (or that parts may be 'aggregated' to form wholes as discussed in Section 3.4). These 'object-oriented' concepts allow software and databases to model the structure of human concepts (thus making them more comprehensible) and they help to minimise redundancies in knowledge—which makes it easier to modify any given body of knowledge without introducing unwanted inconsistencies.

In the database world, object-oriented concepts have been developed in the 'entity-relationship model' [16] and the 'enhanced entity-relationship model' (see [18]) and also in a variety of 'object-oriented databases' (see [10]). (In the remainder of this paper, the entity-relationship model and enhanced entity-relationship model will be referred to collectively as the entity-relationship model.)

In the SP system, all the object-oriented concepts mentioned in the previous paragraphs may be expressed and integrated using SP patterns, as illustrated in the multiple alignment shown in Fig. 9. This is the best alignment found between the set of New patterns {'<capacity> 2000cc </capacity>', '<seats> 4 </seats>' 'spark_plugs'} and a set of Old patterns representing concepts related to motor vehicles. As before, the New patterns appear in column 0 and the remaining columns contain Old patterns, one per column in any order. The whole alignment may be seen to be the result of a process of recognition, with the New patterns representing features of some unknown entity and the Old patterns representing the results of the recognition process.[7]

In this example, the unknown entity is recognized at several different levels of abstraction: as a vehicle, as a car, and more specifically as the car with the registration number 'LMN 888'. These levels of abstraction are represented by different patterns in the alignment:

- The pattern in column 3 represents the class 'vehicle'. At this abstract level, a 'vehicle' is something with a registration number, an engine, steering wheel, seats, and so on, but the details are unspecified.
- Some of that detail is provided by the pattern in column 4 that represents the subclass 'car'. In this example, a car is a vehicle with 4 seats, 4 doors and 4 wheels and a relatively small space for carrying luggage.
- Yet more detail is supplied by the pattern shown in column 1 that represents a specific instance of a car with an identifier ('v4'), a registration number ('LMN 888'), and with a gasoline type of engine with a capacity of 2 litres.

[7] Notice that the order in which the New patterns appear in column 0 of any alignment need not be the same as the order in which they are supplied to the program.

Fig. 9. A multiple alignment created by SP62 showing how class-inclusion hierarchies and part-whole hierarchies may be expressed and integrated within the SP framework.

Other information about the engine in v4 is provided by other patterns in the alignment representing different levels of abstraction:

- The pattern in column 2 represents the general structure of the class of internal combustion engines. At this abstract level, an engine is something with fuel, a 'capacity', some level of 'compression', and a cylinder block, crank shaft, piston and valves.
- More detail is provided by the pattern in column 5 which tells us that, as a gasoline type of engine, it runs on gasoline fuel, that it has a (relatively) low compression and that, in addition to the parts mentioned earlier, it has spark plugs and a carburettor. The main alternative to gasoline-type engines is, of course, the diesel type—not shown in the figure—which runs on diesel fuel, has a (relatively) high compression, and does not need spark plugs or a carburettor.

Readers may wonder why the symbols '<v>' and '</v>' are used in the patterns shown in columns 1, 3 and 4 and why '<e>' and '</e>' appear in columns 1, 2 and 5. These symbols are, in effect, 'punctuation' symbols that are needed to ensure that multiple alignments can be formed according to the principles described in [35,37] and earlier publications.

4.1. Discussion

The multiple alignment concept, as it has been developed in the SP framework, provides a means of expressing all the main constructs associated with object-oriented design:

- *Classes, subclasses and objects*. In Fig. 9, there is a hierarchy of classes from 'vehicle' at the top level (column 2) through 'car' at an intermediate level (column 4) to an individual object ('v4' shown in column 1) at the bottom level. The class 'engine' is also shown at an abstract level (column 3) and at a more concrete level (column 5).
- *Parts and sub-parts*. In our example, the class 'vehicle' has parts such as 'engine', 'steering_wheel', 'seats', and so on, and the 'engine' has parts such as 'cylinder_block', 'crank_shaft', etc. If there was only one type of engine, then all the parts and other attributes of engines could be expressed within the 'vehicle' pattern, without the need for a separate pattern to represent the engine. The reason that a separate pattern is needed—with a corresponding slot in the 'vehicle' pattern—is that there is more than one kind of engine. Another reason for representing the class of engines with separate patterns is that engines may be used in a variety of other things (e.g., boats, planes and generators), not just in road vehicles.
- *Inheritance of attributes*. From the multiple alignment in Fig. 9 we can infer that vehicle 'v4' has a cylinder block, crank shaft, pistons and valves, that the engine has a low compression, that the vehicle has 4 wheels, and so on. None of these 'attributes' are specified in the pattern for v4 shown in column 1. They are 'inherited' from patterns representing other classes, in much the same way as in other object-oriented systems.
- *Cross-classification and multiple inheritance*. The multiple alignment framework supports cross-classification with multiple inheritance just as easily as it does simple class hierarchies with single inheritance. With our 'vehicle' example, it would be easy enough to introduce patterns representing, say, 'military vehicles' or 'civilian vehicles', a classification which cuts across the division of vehicles into categories such as 'car', 'bus', 'van', and so on. In a similar way, vehicles can be cross-classified as 'gasoline_type' or 'diesel_type' on the strength of the engines they contain, as shown in our example.

Within the SP framework, part-whole hierarchies may be—and normally should be—fully integrated with the class-inclusion hierarchies to which they relate. The parts and sub-parts of any class or entity are also the 'attributes' that describe each class (see Section 4.2.1).

4.1.1. Variables, values and types

It should be apparent that, in the SP system, a pair of neighboring symbols like '<fuel>' and '</fuel>' function very much like a 'variable' in a conventional system. By appropriate alignment within a multiple alignment, such a variable may receive a 'value' such as 'gasoline' or 'diesel' in this example. The range of possible values that a given variable may take—the 'type' of the variable—is defined implicitly within any given set of patterns in Old.

4.1.2. Variability of concepts

Column 4 in Fig. 9 shows a car as something with 4 seats, 4 doors and 4 wheels but of course we know that all of these values can vary. Sports cars often have 2 seats and 2 doors, some budget cars have 3 wheels, and a stretch limo may have many more seats and doors. In a more fully developed example, numbers of seats, doors and wheels would be unspecified at the level of 'car' and would be defined in subclasses like those that have been mentioned.

4.1.3. Polymorphism

In object-oriented systems, two or more objects in different classes may respond, each in their own way, to one 'message', in much the same way that the command ''Start training'' issued by the leader of a sports club means running for runners, swimming for swimmers, and so on.

This kind of 'polymorphism' has no direct counterpart in the SP system because it has no concept of 'message' or 'method'. However, its ability to model hierarchies of classes and subclasses implies that any reference to a high-level class is likely to mean a variety of different things at lower levels in the class hierarchy, not unlike polymorphism in ordinary object-oriented systems.

4.2. Comparison between the SP model and other object-oriented systems

As in Section 3.6, the SP model has advantages over object-oriented models in its relative simplicity, expressive power, versatility, and robustness in the face of errors. The following subsections consider a selection of other differences that are somewhat more subtle but are, nevertheless, important.

4.2.1. Parts, attributes and inheritance

In Simula and most object-oriented systems that have come after, there is a distinction between 'attributes' of objects and 'parts' of objects. The former are defined at compile time while the aggregation of parts to form wholes is a run-time process. This means that the inheritance mechanism applies to attributes but not to parts.

In the SP system, this distinction disappears. Parts of objects can be defined at any level in a class hierarchy and inherited by all the lower level. The parts and sub-parts of a class are *also* 'attributes' of the class that may be inherited by lower-level classes or objects. *There is seamless integration of class hierarchies with part-whole hierarchies.*

4.2.2. Objects, classes and metaclasses

By contrast with most object-oriented systems, the SP system makes no formal distinction between 'class' and 'object'. This accords with the observation that what we perceive to be an individual object, such as 'our car', can itself be seen to represent a variety of possibilities such as 'our car taking us on holiday', 'our car carrying the shopping' and so on. A pattern like the one shown in column 1 of Fig. 9 could easily function as a class with vacant slots to be filled at a more specific level by details of the passengers or load being carried, the role the vehicle is playing, the color it has been painted, and so on. This flexibility is lost in systems that do make a formal distinction between classes and objects.

Another consequence of making a formal distinction between objects and classes is that it points to the need for the concept of a 'metaclass':

> If each object is an instance of a class, and a class is an object, the [object-oriented] model should provide the notion of *metaclass*. A metaclass is the class of a class. [10, p. 43].

It is true that this construct is not provided in most object-oriented database systems but it has been introduced in some artificial intelligence systems so that classes can be derived from metaclasses in the same way that objects are derived from classes. Of course, this logic points to the need for 'metametaclasses', 'metametametaclasses', and so on without limit.

Because the SP system makes no distinction between 'object' and 'class', there is no need for the concept of 'metaclass' or anything beyond it. All these constructs are represented by SP patterns.

4.2.3. The entity-relationship model with a relational database

With minor variations, the entity-relationship model has become the mainstay of data processing applications for business and administration. Diagrammatic representations of entities and relationships (like the one shown in Fig. 6) are normally implemented with a relational database and there are software tools to do the translation, hiding many of the details. Since this combination of entity-relationship model and relational database has come to be so widely used, it will be the focus of our discussion here.

A table can be used to represent a class, with the columns (fields) representing the attributes of the class and the rows representing individual instances of the class. Each class or subclass in a class hierarchy can also be

represented by a table but in this case it is necessary to provide additional fields so that the necessary connections can be made. For example, the class of 'staff' in a company may be represented by a table like the one shown in Table 1 and separate tables may be created for each of the subclasses 'manager', 'supervisor' and 'assistant', each of these with columns relevant to the particular subclass but not for other subclasses. In addition, each of the tables for the subclasses needs a column such as 'Staff Number' so that the record of an individual in any one subclass can be connected to the corresponding record in the superclass. Similar principles apply to the division of concepts into parts and sub-parts.

This system works quite well for many applications but it has a number of shortcomings compared with the SP system:

- Using tables to represent classes means that the description of a class must always take the form of a set of variables corresponding to the fields in the table. In the SP system, it is possible to describe a class using any combination of variables and literals, according to need. It is, for example, possible to record that a vehicle has a steering wheel (as in column 2 of Fig. 9) without any implication that there may be alternative kinds of steering wheel to be recorded in a field with that name. It is also possible to provide a verbal description of any class, something that is outside the scope of the relational model.
- Using tables to represent classes means that the record for each specific object must have start and end tags for every field in the table regardless of whether or not all the fields are used. In the SP system, start and end tags are only needed for the fields that contain a value in the record for any specific instance.
- The SP system allows the description of class hierarchies and part-whole hierarchies to be separated from the description of individual members of those hierarchies. By contrast, the use of tables to represent class hierarchies and part-whole hierarchies means that the structure of these hierarchies must be reproduced, again and again, in every instance. Using tables to represent either kind of hierarchy means that information that is specific to any one individual is fragmented and must be pieced together using keys. In the SP system, by contrast, information that is specific to any one individual can always be represented with a single pattern. The SP system provides for the seamless integration of class hierarchies and part-whole hierarchies in a way that cannot be achieved using tables.

5. The hierarchical and network models

Although the hierarchical and network models for databases have fallen out of favour in ordinary data processing applications, the network model has seen a revival, first with the development of the hypertext concept and then more dramatically with the application of that concept to the World Wide Web. The hierarchical model is the mainstay of hierarchical file systems and finds niche applications in directories of various kinds.

In the SP system, any network or hierarchy can be represented using connections between patterns like the ones shown here:

```
0     a1     b1      a2     0
      |      |       |
1 <A> a1 <B> |  </B> a2 </A> 1
           |  |   |
2        <B> b1 </B>        2.
```

The basic idea is that one pattern, 'A', may contain the start and end symbols of another pattern, 'B', so that the two patterns can be connected within a multiple alignment, as shown. In effect, the pair of symbols '<B> </B>' in the 'A' pattern (row 1) are a 'reference' to the 'B' pattern (row 2). With this simple device, it is possible to link patterns to form hierarchies and networks of any complexity, including class-inclusion hierarchies, part-whole hierarchies and their integration as shown in Fig. 9.

Any one pattern may appear two or more times within a multiple alignment, as noted in Section 2.2.1 (see also [35] and earlier publications). This allows recursive structures to be modelled (see [33]).

Where the full versatility of this scheme is not needed, it is also possible to create simpler networks and hierarchies by means of end-to-end matches between patterns. To illustrate this idea, the hierarchy shown

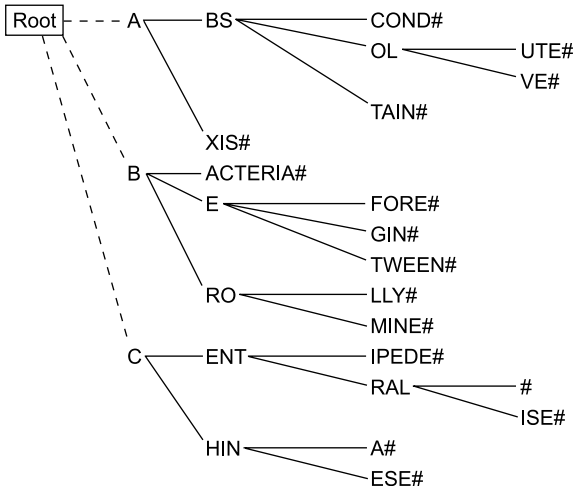


Fig. 10. A 'discrimination network' for the recognition of words. The character '#' serves to mark the right-hand ends of words.

```
Root A 1
     1 B S 2
          2 C O N D #
          2 O L 3
               3 U T E #
               3 V E #
          2 T A I N #
     1 X I S #
Root B 4
     4 A C T E R I A #
     4 E 5
          5 F O R E #
          5 G I N #
          5 T W E E N #
     4 R O 6
          6 L L Y #
          6 M I N E #
Root C 7
     7 E N T 8
          8 I P E D E #
          8 R A L 9
               9 #
               9 I S E #
     7 H I N 10
          10 A #
          10 E S E #
```

Fig. 11. A set of SP patterns representing the network shown in Fig. 10. The indentation is intended as an aid to readers to see the correspondences between the patterns in this figure and the network shown in Fig. 10.

in Fig. 10 (a 'discrimination network' for the recognition of words) may be represented by the set of SP patterns shown in Fig. 11, while Fig. 12 shows how these patterns may be aligned in the recognition of the word 'absolute'.[8] This alignment is one of the two best alignments found by SP62 with the set of patterns from Fig. 11 in Old and the pattern 'A B S O L U T E #' in New.[9]

[8] The character '#' in these examples serves to mark the right-hand ends of words.

[9] The other alignment found with the same high score is the same as this alignment except that the order of the Old patterns across rows 1 to 4 is different. This difference in ordering has no significance and the two alignments are equivalent.

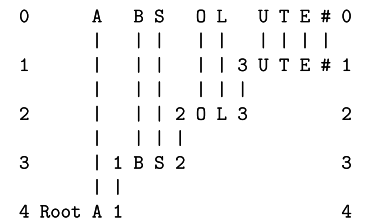


Fig. 12. Recognition of a word using a set of SP patterns representing a discrimination network: the best multiple alignment found by SP62 with the New pattern 'A B S O L U T E #' and Old patterns as shown in Fig. 11.

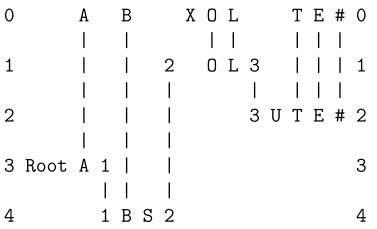


Fig. 13. Recognition of a mis-spelled word using a set of SP patterns representing a discrimination network: the best multiple alignment found by SP62 with the New pattern 'A B X O L T E #' and Old patterns as shown in Fig. 11.

5.1. Comparison between the SP model and the hierarchical and network models

The hierarchical and network models are, arguably, just as simple as the SP model but they lack the expressive power of that model—the ability to emulate a range of other representational schemes—and they lack the versatility of the SP model to perform the kinds of AI functions outlined in Section 6. As with the relational and object-oriented models (Sections 3.6 and 4.2), the hierarchical and network models, as they are usually implemented, lack the ability of the SP model to cope with errors of omission, commission or substitution.

To illustrate the last point, Fig. 13 shows how the mis-spelled word 'A B X O L T E #' may be recognised amongst the patterns from Fig. 11, despite one error of substitution ('X' substituted for 'S') and one error of omission (the letter 'U'). Any ordinary discrimination network will fail unless all the elements of the input pattern are correct.

6. The SP model and aspects of intelligence

This section briefly reviews aspects of intelligence that have been shown to fall within the scope of the SP system. The main points of difference between the SP system and other AI systems are summarised. Readers are referred to [35,37] and other sources cited therein for more detail about capabilities of the system that are outlined below. A relatively full discussion of those capabilities in relation to intelligent databases is presented in [38].

That the system can demonstrate certain kinds of capability does not, of course, mean that its performance in a given area is perfect or comprehensive. Further development is needed in all of the areas to be described, and of course some areas are further advanced than others.

With that qualification, the main capabilities of the SP system are these:

- *Representation of knowledge*. Although all kinds of knowledge in the SP system are represented as 'flat' patterns, the way in which the system builds multiple alignments means that these flat patterns can be used, as we have seen, to represent class-inclusion hierarchies, part-whole hierarchies, and simpler kinds of networks and trees. They can also be used to represent context-free and context-sensitive grammars, if-then rules, and other styles of representation used in AI.

Compared with 'sub-symbolic' kinds of representation used in artificial neural networks, this kind of 'symbolic' representation has the advantage from a user's perspective that knowledge structures are explicit and open to inspection. When patterns are used to model such things as class-inclusion hierarchies and part-whole hierarchies, there is the additional advantage that these structures appear to users to be natural and easy to understand.

Using one simple format for diverse kinds of knowledge facilitates their seamless integration.

- *Fuzzy pattern recognition and recognition at multiple levels of abstraction*. The dynamic programming at the heart of the SP system [31] provides for fuzzy recognition of patterns and objects: things can be recognized via a global best-match which is not unduly sensitive to errors of omission, commission or substitution.
  As we saw in Section 4, when the system is provided with Old patterns representing one or more class-inclusion hierarchies, it is able to recognize things at multiple levels of abstraction.
- *Best-match and semantic forms of information retrieval*. As we saw in Section 3, the SP system provides a model for information retrieval. As with recognition, retrieval of information does not depend on there being an exact match between patterns.
  Given that the system has been supplied with Old patterns representing associations between synonyms or near-synonyms, it is able to retrieve information by meanings and not purely by direct matches between a query pattern and one or more targets. Thus, for example, 'house' in a query may retrieve a pattern showing that 'house' is associated with 'dwelling' and this pattern may then serve to retrieve patterns containing information about dwellings.
- *Probabilistic reasoning*. In any multiple alignment, any Old symbol that is not matched with a New symbol represents an inference that may be drawn from the alignment. This is the key to reasoning in the SP system. As an example, we may infer from Fig. 9 that the car that has been recognized uses gasoline fuel, that it has 4 doors and that the engine has valves. Probabilities may be calculated for all such inferences using information about frequencies that is associated with each pattern (Section 2.1).
  The SP framework supports a variety of kinds of probabilistic reasoning, including probabilistic 'deduction', abduction, reasoning with probabilistic decision networks and trees, reasoning with 'rules', chains of reasoning, non-monotonic reasoning and reasoning with default values, 'explaining away', causal diagnosis, and reasoning that is not supported by evidence. The building of multiple alignments provides the key to complex forms of reasoning.
  Relevant examples and discussion may be found in [33] and [37, Chapter 7].
- *Exact forms of reasoning and calculation*. Although the SP system is primarily probabilistic, it can be constrained to work in the 'exact' style of classical logic and mathematics. This aspect of the system and how it may be used for traditional kinds of deduction and calculation are discussed in [37, Chapter 10].
- *Analysis and production of natural language*. As we saw in Section 2.2.2, SP patterns may be used to represent grammatical rules, and the syntactic parsing of a sentence in natural language may be modelled by the building of multiple alignments. Without any modification, the same framework may be used for the representation of 'meanings', the integration of syntax and semantics, the derivation of meanings from sentences, and the production of sentences from meanings (see [34] and [37, Chapter 5]).
- *Planning and problem solving*. Given appropriate patterns, the SP system may do such things as finding a route between two places or solving geometric analogy problems [37, Chapter 8].
- *Unsupervised learning*. The acquisition of New patterns and their compression into Old patterns provides a model for such things as the unsupervised learning of a natural language or the building of non-linguistic conceptual structures. The SP70 computer model (see [36] and [37, Chapter 9]) is able to induce simple grammars from appropriate input but more work is needed to realize the full potential of the model.

6.1. Relationship with other artificial intelligence systems

It would take us too far afield to attempt a detailed comparison with artificial intelligence systems in all the areas considered above. As an attempt to integrate ideas across a wide area, the SP system naturally has points of similarity with many existing systems, but at the same time, it has its own distinctive features.

Chief amongst these is the remarkable simplicity of the system combined with its very wide scope, much wider than the great majority of artificial intelligence systems, with the possible exception of unified theories of cognition such as Soar [19,27] and ACT-R [6]. Like those two systems, the development of the SP system was inspired in part by the writings of Allen Newell, putting the case for greater breadth and depth in theories of cognition [23,24].

Unlike hybrid systems, of which there are many, the SP system is not merely a conjunction of two or more different systems, combining their capabilities and also their complexities. The SP system is the result of a radical rethink of concepts in artificial intelligence and beyond, achieving integration in a radically simplified structure. The result is a conceptual framework with distinctive features of which the main ones are

- *All* kinds of knowledge are represented with flat patterns.
- *All* kinds of processing is achieved by compression of information by the matching and unification of patterns.
- The use of a modified version of the concept of multiple alignment as a vehicle for the recognition of patterns, information retrieval, probabilistic and exact forms of reasoning, and other aspects of intelligence.

7. Developing the system

The SP computer models (SP62 and SP70) are good enough to demonstrate what can be done with the system but fall short of what would be needed for applications in industry, commerce or administration. This section considers briefly how the SP concepts that have been developed to date may be translated into a practical system.

7.1. Computational complexity

Like many other problems in artificial intelligence, the building of 'good' multiple alignments with realistic data sets is intractable if one wishes to obtain answers that are theoretically ideal. But for many purposes, we need only to obtain answers that are 'good enough'. With this relaxation of requirements, it is often possible to achieve dramatic reductions in time complexity or space complexity or both.

In a serial processing environment, the time complexity of the SP62 model is polynomial and within the bounds of what is normally regarded as tolerable. The space complexity in serial or parallel environments is O(m), where m is the size of the set of Old patterns, in bits. Other SP models are similar. As outlined below, it should also be possible to improve running times and time complexity by the application of parallel processing.

It is worth noting that, at the heart of the SP system is a process for finding partial matches between patterns that is similar in its effects to the matching processes in Google. That system provides an existence proof of the speed that can be achieved in searching for good full and partial matches between a target ('query') pattern and other patterns within a large volume of stored data.

7.2. Parallel processing

Significant improvements in time complexity are to be expected if the system can be developed with the benefit of parallel processing. And of course, parallel processing brings the additional benefit of faster processing in absolute terms and, with suitable design, greater robustness in the face of system failures. Parallel processing is now a recognised requirement to meet the high computational demands of large-scale databases [1,2] and large-scale applications in artificial intelligence.

The SP machine does not necessarily have to be developed in silicon. One futuristic possibility is to exploit the potential of organic molecules such as DNA or proteins—in solution—to achieve the effect of high-parallel pattern matching. This kind of 'molecular computation' is already the subject of much research (see, for example, [3,4]) and techniques of that kind could, conceivably, form the basis of a high-parallel SP machine.

Another possibility is to use light for the kind of high-speed, high-parallel pattern matching that is needed at the heart of the SP machine. Apart from its speed, an attraction of light in this connection is that light rays can cross each other without interfering with each other, eliminating the need to segregate one stream of signals from another (see, for example, [17,21]).

On relatively short time scales, a silicon version of the SP machine would probably be the easiest option and it may be developed in at least four different ways:

- It should be feasible to design new hardware for the kind of high-parallel pattern matching that is needed.
- Given that SIMD and MIMD high-parallel computers are already in existence, an alternative strategy is to create the SP machine as a software virtual machine running on one of these platforms.
- An existing high-parallel database system (see, for example, [26,22]) may be modified to support the SP model. Other models may, of course be retained alongside the SP model.
- The system may be developed using low-cost computers connected together in a local area network or even a wide area network. Systems like Google have been developed in this way and they already provide high-speed pattern matching of a kind that may be adapted for use within a software implementation of the SP machine.

7.3. User interface

A graphical user interface to the SP system is needed for the input of data and queries, for the setting of parameters, and for viewing data and results.

For some applications there may be a need to provide an SQL-like query language. As indicated in Section 3.2, it seems likely that this may be achieved within the SP framework by means of an appropriate set of SP patterns—but the details of how that may be done would need investigation.

7.4. Hybrid solutions

In the development and application of information systems, it is rarely possible to introduce a new model and simultaneously discard all pre-established models. There is normally a transition phase, which may be very prolonged, where two or more models coexist as alternatives for different applications or are used in some combination, according to need. Notwithstanding the remarks made in Section 6.1 about hybrid systems, there may well be a case during the transition phase to create hybrids between the SP system and other kinds of system. This may be done in at least three different ways:

- The system may be used in conjunction with existing languages and with arithmetic or mathematical functions, in very much the same way that the relational database model is standardly used in conjunction with these non-relational languages and functions in many data processing applications.
- An SP database system may serve as a framework within which existing applications may be embedded, in much the same way that a relational or object-oriented database—or, indeed, an hierarchical file system—may contain pointers to executable files of many different kinds (see, for example, [14]).
- Pending a fuller development of the system for such applications as data mining, there is no reason why existing data-mining techniques should not be used with an SP database.

8. Conclusion

The SP model is an alternative to existing database models that offers significant benefits compared with those models. Within the SP framework it is possible to represent knowledge in a simple, comprehensible and versatile format that facilitates the seamless integration of different kinds of knowledge. The building of multiple alignments provides a key to several aspects of intelligence, including fuzzy recognition of patterns, best-match retrieval of information, probabilistic and exact kinds of reasoning, unsupervised learning,

planning, problem solving and others. And the system has an ability to function effectively despite errors in its input data.

The main sections of this paper have sought to show how the SP system can function like established database models when that is required. Although the treatment has not been exhaustive, the paper has discussed how the SP framework may emulate the main types of database model including:

- The *relational model* and how tables may be represented in the manner of XML, how information may be retrieved in the manner of query-by-example, how information may be retrieved from a 'join' of two or more tables, the way in which a query language such as SQL may be modelled within the system, how the SP model relates to the concept of 'database schema', how the concept of 'aggregation' may be modelled within the system, and how the SP system relates to the concepts of 'database schema', 'referential integrity' and 'normalisation'.
- *Object-oriented models* and how the system may emulate such things as database 'objects', hierarchies of classes and subclasses (including cross-classification), hierarchies of parts and sub-parts of objects and their integration with class hierarchies, inheritance of attributes (including 'multiple' inheritance), and concepts of 'variable', 'value' and 'type'. Because the SP system makes no formal distinction between 'object' and 'class', there is no need for such concepts as 'metaclass', 'metametaclass' and so on. Also discussed is the use of the relational model as a vehicle for the entity-relationship model, problems associated with that combination, and solutions to those problems provided by the SP system.
- The *hierarchical and network models* including the way in which the SP system may emulate a discrimination network. Also discussed in this connection is the contrast between the robustness of the SP system in the face of errors in its input data and the way in which traditional discrimination networks require every detail to be correct.

Developing the system for application as an intelligent database is likely to involve three main phases or strands of work (which may overlap each other):

- More work is required in developing the SP framework itself, especially those aspects of the framework concerned with the unsupervised processing of raw data, semi-structured data or badly structured data to create well-structured knowledge. A better understanding is needed of how the system may be used to function in the manner of a conventional programming language and how things like arithmetic may be modelled within the system. A related strand of work is understanding how the functionality of a language like SQL may be realised within the system.
- Although the full framework of the SP system is still not fully developed, the parts of the system that build multiple alignments—corresponding to the SP62 computer model—are relatively mature and may be applied to a range of tasks including pattern recognition, information retrieval, various kinds of reasoning, natural language processing, and more. A programme of work is needed to translate this partial model into a fully fledged system that can be used for real applications. As was outlined above, the searching and matching processes in the system are likely to benefit from the application of parallel processing by improving the time complexity of the model as well as absolute running times. An interesting possibility is to develop the SP system using the searching and matching processes provided in Google. Another interesting possibility is to use light for the high-speed matching of patterns which lies at the heart of the SP system.
- Pending full development of the abstract system, there is likely to be a case for developing hybrids between the SP model and established systems for arithmetic, ordinary programming and the like, in much the same way that computers used to include maths co-processors and relational databases typically incorporate non-relational programming languages.

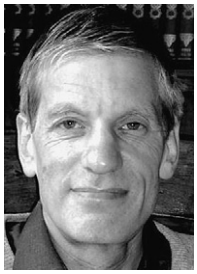
Acknowledgement

I am very grateful to Thomas Connolly and to anonymous referees for constructive comments on earlier versions of this paper. The responsibility for all errors and oversights is, of course, my own.

References

[1] M. Abdelguerfi, S. Lavington (Eds.), Emerging Trends in Database and Knowledge-base Machines: The Application of Parallel Architectures to Smart Information Systems, IEEE Computer Science Press, Los Alamitos, CA, 1995.
[2] M. Abdelguerfi, K.-F. Wong, Parallel Database Techniques, IEEE Computer Science Press, Los Alamitos, CA, 1998.
[3] L.M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024.
[4] L.M. Adleman, Computing with DNA, Scientific American 279 (2) (1998) 54–61.
[5] L. Allison, C.S. Wallace, The posterior probability distribution of alignments and its application to parameter estimation of evolutionary trees and to optimization of multiple alignments, Journal of Molecular Evolution 39 (1994) 418–430.
[6] J.R. Anderson, C.J. Lebiere, The Atomic Components of Thought, Lawrence Erlbaum, Magwah, NJ, 1998.
[7] F. Attneave, Some informational aspects of visual perception, Psychological Review 61 (1954) 183–193.
[8] H.B. Barlow, Trigger features, adaptation and economy of impulses, in: K.N. Leibovic (Ed.), Information Processes in the Nervous System, Springer, New York, 1969, pp. 209–230.
[9] H.B. Barlow, The exploitation of regularities in the environment by the brain, Behavioural and Brain Sciences 24 (4) (2001) 602–607, 748–749.
[10] E. Bertino, B. Catania, G.P. Zarri, Intelligent Database Systems, Addison-Wesley, Harlow, 2001.
[11] G.M. Birtwistle, O.-J. Dahl, B. Myhrhaug, K. Nygaard, Simula Begin. Studentlitteratur, Lund, 1973.
[12] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modelling Language User Guide, Addison-Wesley, Reading, MA, 1999.
[13] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language (XML) 1.0, 3rd ed. Technical report, World Wide Web Consortium, 2004. W3C recommendation, 4 February 2004. Available from: www.w3.org/TR/2004/REC-xml-20040204/.
[14] F. Cariño, W. Sterling, Parallel strategies and new concepts for a petabyte multimedia database computer, in: Parallel Database Techniques [2], pp. 139–164.
[15] N. Chater, The search for simplicity: a fundamental cognitive principle? Quarterly Journal of Experimental Psychology A 52 (2) (1999) 273–302.
[16] P.P. Chen, The entity-relationship model—toward a unified view of data, ACM Transactions on Database Systems 1 (1) (1976) 9–36.
[17] O.H. Cho, R.M. Colomb, Associative random access machines and data-parallel multiway binary-search join, Future Generation Computer Systems 13 (6) (1998) 451–467.
[18] T. Connolly, C. Begg, Database Systems, Addison-Wesley, London, 2002.
[19] J.E. Laird, A. Newell, P.S. Rosenbloom, Soar: an architecture for general intelligence, Artificial Intelligence 33 (1987) 1–64.
[20] M. Li, P. Vitányi, An Introduction to Kolmogorov Complexity and Its Applications, Springer Verlag, New York, 1997.
[21] A. Louri, J.A. Hatch, An optical associative parallel processor for high-speed database-processing—theoretical concepts and experimental results, Computer 27 (11) (1994) 65–72.
[22] T. Mahapatra, S. Mishra, Oracle Parallel Processing, O'Reilly, UK, 2000.
[23] A. Newell, You can't play 20 questions with nature and win: projective comments on the papers in this symposium, in: W.G. Chase (Ed.), Visual Information Processing, Academic Press, New York, 1973, pp. 283–308.
[24] A. Newell (Ed.), Unified Theories of Cognition, Harvard University Press, Cambridge, MA, 1990.
[25] R.C. Oldfield, Memory mechanisms and the theory of schemata, British Journal of Psychology 45 (1954) 14–23.
[26] J. Page, A study of a parallel database machine and its performance: the NCR Teradata DBC-1012, Lecture Notes in Computer Science 618 (1992) 115–137.
[27] P.S. Rosenbloom, J.E. Laird, A. Newell, The Soar Papers: Readings on Integrated Intelligence, MIT Press, Cambridge, MA, 1993.
[28] D. Sankoff, J.B. Kruskall, Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparisons, Addison-Wesley, Reading, MA, 1983.
[29] R.J. Solomonoff, A formal theory of inductive inference. Parts I and II, Information and Control 7 (1964) 1–22, 224–254.
[30] J.G. Wolff, Language acquisition data compression and generalization, Language & Communication 2 (1982) 57–89. Available from: www.cognitionresearch.org.uk/lang_learn.html#wolff_1982.
[31] J.G. Wolff, A scaleable technique for best-match retrieval of sequential information using metrics-guided search, Journal of Information Science 20 (1) (1994) 16–28. Available from: www.cognitionresearch.org.uk/papers/ir/ir.htm.
[32] J.G. Wolff, Computing as information compression by multiple alignment, unification and search, Journal of Universal Computer Science 5 (11) (1999) 777–815. Available from: http://arxiv.org/abs/cs.AI/0307013.
[33] J.G. Wolff, Probabilistic reasoning as information compression by multiple alignment, unification and search: an introduction and overview, Journal of Universal Computer Science 5 (7) (1999) 418–462. Available from: http://arxiv.org/abs/cs.AI/0307010.
[34] J.G. Wolff, Syntax, parsing and production of natural language in a framework of information compression by multiple alignment, unification and search, Journal of Universal Computer Science 6 (8) (2000) 781–829. Available from: http://arxiv.org/abs/cs.AI/0307014.
[35] J.G. Wolff, Information compression by multiple alignment, unification and search as a unifying principle in computing and cognition, Artificial Intelligence Review 19 (3) (2003) 193–230. Available from: http://arxiv.org/abs/cs.AI/0307025.
[36] J.G. Wolff, Unsupervised grammar induction in a framework of information compression by multiple alignment, unification and search, in: C. de la Higuera, P. Adriaans, M. van Zaanen, J. Oncina (Eds.), Proceedings of the Workshop and Tutorial on Learning Context-Free Grammars, pp. 113–124, 2003. This workshop was held in association with the 14th European Conference on Machine Learning and the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2003), September 2003, Cavtat-Dubrovnik, Croata. Available from: http://arxiv.org/abs/cs.AI/0311045.

[37] J.G. Wolff, Unifying computing and cognition: the SP theory and its applications, CognitionResearch.org.uk, Menai Bridge, 2006. This is an e-book, ISBN 0-9550726-0-3, available from Amazon.com and other distributors. A print edition of the book, ISBN 0-9550726-1-1, is due out soon. For up-to-date information see www.cognitionresearch.org.uk/sp.htm#BOOK.
[38] J.G. Wolff, Aspects of intelligence in an 'SP' intelligent database system, in: Z. Ma (Ed.), Intelligent Databases: Technologies and Applications. Idea Group Inc., Hershey, PA, forthcoming.



Gerry Wolff is Director of CognitionResearch.org.uk. Previously, he has held academic posts in the School of Informatics, University of Wales, Bangor, and the Department of Psychology, University of Dundee, a Research Fellowship with IBM in Winchester, UK, and he has worked for four years as a Software Engineer with Praxis Systems in Bath, UK. His first degree is in Natural Sciences (Psychology) from Cambridge University and his PhD is in Cognitive Science from the University of Wales. He is a Chartered Engineer and a Member of the British Computer Society (Chartered IT Professional). Since 1987 his research has focussed on the development of the SP theory. Previously his main research interests were in developing computer models of language learning. He has numerous publications in a wide range of journals, collected papers and conference proceedings.