Int. J. Man–Machine Studies (1988) **29,** 407–427

Effects of computer programming experience on network representations of abstract programming concepts

NANCY J. COOKE AND ROGER W. SCHVANEVELDT†

Department of Psychology, Rice University, Houston, Texas 77251, USA and
† Computing Research Laboratory and Department of Psychology, New Mexico State University, USA

(Received 6 August 1987)

The cognitive organization of a set of abstract programming concepts was investigated in subjects who varied in degree of computer programming experience. Relatedness ratings on pairs of the concepts were collected from naive, novice, intermediate, and advanced programmers. Both individual and group network representations of memory structure were derived using the Pathfinder network scaling algorithm. Not only did the four group networks differ, but they varied systematically with experience, providing support for the psychological meaningfulness of the structures. Additionally, an analysis at the conceptual level revealed that the four groups differed in the way concepts were represented. Furthermore, this analysis was used to classify concepts in the naive, novice, and intermediate networks as well-defined or misdefined. The identification of semantic relations corresponding to some of the links in the networks provided further information concerning differences in programmer knowledge at different levels of experience. Applications of this work to programmer education and knowledge engineering are discussed.

Introduction

The investigation of cognitive differences between experts and novices is important to those interested in education or training programs that attempt to make experts out of novices or select individuals for their potential expertise (e.g. Schvaneveldt, Durso, Goldsmith, Breen, Cooke, Tucker & DeMaio, 1985) and those concerned with the development of expert knowledge-based systems that rely heavily on expert knowledge (Cooke, 1985; Dreyfus & Dreyfus, 1987). By definition, experts' performance on tasks within their domain of expertise is superior to that of novices. Specific differences between experts and novices have been investigated in tasks requiring recall, categorization, relatedness judgements, and problem solving. In general, the results of these studies have indicated that not only do experts possess more knowledge than novices, but any knowledge that is shared by the two groups is organized differently in memory.

Researchers who have investigated expertise in the domains of chess, bridge, and Go, found that experts were able to recall more information than novices when the to-be-recalled material was presented in a meaningful fashion. On the other hand,

Please send correspondence to Nancy J. Cooke, P.O. Box 1892, Department of Psychology, Rice University, Houston, TX 77251, U.S.A.

0020-7373/88/090407 + 21$03.00/0

there was no advantage of expertise when the material was randomly arranged (Chase & Simon, 1973; Engle & Bukstel, 1978; Reitman, 1976). Because the experts in these studies did not have a greater memory capacity than the novices, their recall advantage was attributed to superior pattern recognition skills and chunking. In other words, experts, by virtue of domain-specific information stored in memory, were able to recognize quickly chunks of related items and hold the chunks as single units in short-term memory.

Several studies have been directed at uncovering the basis of experts' chunks. That is, given that a characteristic of expertise is chunking of domain-related stimuli and that the chunking process is driven by information stored in memory, then it is of interest to study the organization of this information in memory. In general, this issue has been addressed by contrasting the judgements (or memory representations derived from judgements) of experts and novices. Differences in the way experts and novices categorize their domain have been found in studies requiring subjects to sort physics problems (Chi, Feltovich & Glaser, 1981), math problems (Schoenfeld & Herrmann, 1982), and childhood disorders (Murphy & Wright, 1984). These effects of expertise on categorical judgements have supported the view that experts and novices differ in the way information is organized in memory. Further support has been provided by a study on fighter pilot expertise in which network and spatial memory representations of instructor and undergraduate trainee pilots were generated from relatedness ratings of pairs of flight-related concepts (Schvaneveldt *et al.*, 1985*b*). Results indicated that the representations of the two groups of pilots were distinct. In fact, individual pilots could be accurately classified as instructors or trainees based on their cognitive representations.

Increased interest in human–computer interaction has led to research on the differences between expert and novice computer users. Much of this research has focused on computer programmers, a specific class of computer users. In general, results have indicated that, like experts and novices in other domains, programming experts and novices organize programming knowledge differently.

The recall results of Chase & Simon (1973) and others have been replicated in the domain of programming using coherent and shuffled FORTRAN (Shneiderman, 1976) or ALGOL (McKeithen, Reitman, Rueter & Hirtle, 1981) programs as stimuli. Increased experience resulted in superior recall performance for the coherent programs, but there was no effect of experience for the shuffled programs. Therefore, programming experts appeared to benefit from the ability to organize and chunk lines of code in coherent programs.

Several studies have addressed expert and novice differences in the organization of programming knowledge. In a second experiment, McKeithen *et al.* (1981) had subjects recall ALGOL reserved words in a multi-trial cued recall task and derived tree structures from the recall protocols to represent the underlying cognitive organization of the reserved words. Experts organized the words based on their meaning in the ALGOL program, whereas novices organized the words according to mnemonics or surface features. McKeithen *et al.* (1981) also noted that experts' tree structures were quite similar to each other, whereas novices' structures were more variable, suggesting that programming expertise was not completely idiosyncratic to individual experts.

In a similar study, Adelson (1981) had expert and novice programmers recall lines of a PPL program. Multidimensional scaling representations of conceptual structures

were generated from the recall order. Results indicated that, in comparison to novices, experts recall more, chunked more information together into a single unit, and agreed more with each other on recall order. In addition, experts organized the code according to programs or routines, whereas novices organized the code according to syntax or surface features. In another study, Adelson (1984) found that experts represented programs in memory at an abstract level of operations, whereas novices' representations focused on concrete aspects of the program such as the methods used to accomplish the operations. Adelson (1985) later investigated experts' categorization of abstract programming concepts and found evidence for a basic level category and prototypicality differences within the category.

Soloway, Ehrlich & Bonar (1982) employed a cloze test technique to investigate programming knowledge in the form of high level plans. Subjects were required to fill in the blank of a Pascal program with a line of Pascal code. Differences between experts and novices were evident in the line of code that was generated. Also, experts' answers were less variable than those of novices.

The findings from these studies are similar and can be summarized as follows: (1) expert programmers recall more of a coherent program than novices, and this effect has been attributed to the experts' ability to chunk lines of code; (2) experts organize programming information according to deep structure (i.e. routines, meaning, or operations), whereas novices organize programming information according to surface structure (i.e. syntax, mnemonics, or methods); and (3) judgements or cognitive representations of a group of experts are less variable than those of a group of novices.

Studies on programming experience have also had several methodological features in common. More specifically, there has been a tendency to use individuals with some programming experience as the least experienced programmers, consequently ignoring naive programmers with no programming experience whatsoever (but see Shneiderman, 1976). The cognitive structures of individuals prior to exposure to domain-related material might reveal various preconceived notions, misconceptions, or prior knowledge that would be of interest to those interested in education. Consequently, naive programmers were included as subjects in this study, along with novice, intermediate, and advanced programmers.

In addition, stimuli used in research on programming expertise have been either lines of programming code or reserved programming words and consequently are highly specific to a programming language (but see Adelson, 1985). Thus some of the cognitive structures that have been observed might be specific to the particular programming language that was used in the study. For instance, although Adelson (1984) found that experts represented PPL programs at an abstract level of operations, the basis of the organization might be quite different for a language such as APL which contains many high-level constructs or operations. The use of abstract programming concepts that are not related to particular programming languages, but that are basic to an understanding of programming, might aid in tapping into what Shneiderman (1980) refers to as "semantic knowledge". According to Shneiderman's (1980) syntactic/semantic model, syntactic knowledge is specific to each programming language, is precise, and easily forgotten. On the other hand, semantic knowledge is not language-specific, but has to do with general programming concepts. In order to address this issue, a set of abstract programming concepts was used as stimuli in this study.

In addition to the use of naive programmers and a set of abstract programming concepts, empirically derived cognitive networks were employed in this study to investigate cognitive differences with variations in programming experience. Recently, an algorithm (Pathfinder) has been developed that generates network representations (PFNETs) of a set of items (Dearholt, Schvaneveldt & Durso, 1985; Schvaneveldt, Durso & Dearholt, 1985). The items represented in the network can take the form of abstract concepts, events, conditions, actions, or individuals. The PFNETs are generated from proximity estimates of pairs of these items. For instance, proximity estimates for a set of concepts can be obtained by having subjects judge the pairwise relatedness of the concepts. It is assumed that the relatedness ratings provide an estimate of distance between concepts in memory. Given the proximity estimates, the Pathfinder algorithm produces a network in which items are represented as nodes and relations between items are represented as links between the nodes. A weight corresponding to the strength of the relationship between two nodes is associated with each link and reflects the distance between the nodes (see Schvaneveldt *et al.*, 1985*a*, for additional details).

The Pathfinder network scaling technique was chosen for this research because it provides information about cognitive structure that seems particularly applicable to the investigation of programming expertise. That is, the PFNETs generated by Pathfinder are not limited to hierarchical relations, as are structures generated by hierarchical cluster analysis. Also, whereas the dimensions of multidimensional scaling representations reveal global relations among concepts, PFNETs provide information about local relations among the concepts and, consequently, suggest specific relations or misconceptions.

Additionally, although Pathfinder is a relatively new scaling technique, it has been applied to numerous domains with promising results. For instance, Pathfinder has been used to elicit and represent knowledge from users of computer systems in order to improve the UNIX help facility (McDonald, Dearholt, Paap & Schvaneveldt, 1986) and to organize a menu system in a cockpit (Roske-Hofstrand & Paap, 1986). In such applications, Pathfinder can be used to organize a menu system so that for each node (menu panel), all linked nodes (other related panels) can be accessed in one step. Pathfinder also has several advantages as a knowledge elicitation tool for expert systems (Cooke & McDonald, 1986; 1987). The methodology requires experts to make simple relatedness judgements as opposed to traditional knowledge engineering interviews in which the expert has to introspect and verbally report on details of his knowledge.

Finally, results from several studies have shown that Pathfinder representations are psychologically meaningful and in some cases are superior to multidimensional scaling representations. More specifically, Cooke, Durso & Schvaneveldt (1986) compared Pathfinder representations to multidimensional scaling representations in terms of their ability to account for recall performance. Results indicated that PFNETs were more predictive of free recall order than multidimensional scaling representations and that lists organized according to the PFNET structure were easier to learn than those organized according to the multidimensional scaling structure.

In general, the Pathfinder scaling technique seems to be a useful alternative to hierarchical clustering and multidimensional scaling, and results of empirical studies

have supported its psychological validity. Therefore, the purpose of this study was to investigate differences in cognitive organization, as represented by PFNETs, with variations in programming experience and to extend previous findings to deal with naive programmers and stimuli that are not specific to a programming language.

Study 1

METHOD

Subjects

A total of 40 subjects (10 in each of four groups) participated in this study. The four groups were defined as follows: (1) advanced—3 or more years of programming experience; (2) intermediate—1 to 3 years of programming experience; (3) novice—up to 1 year of programming experience; and (4) naive—no programming experience. Programming experience was defined for the purposes of this study as continuous programming experience which could include class work, job experience, and programming done at home. The majority of the subjects were introductory psychology students at New Mexico State University who participated in the study in order to partially fulfill a research requirement. Some of the intermediate and advanced subjects were volunteer graduate sutdents and faculty from the computer science and psychology departments.

Each subject completed a questionnaire concerning the specifics of his or her programming experience. Advanced programmers averaged 7·7 years (range = 3·5 to 15·0) of programming experience and had used an average of 6·7 (range = 4·0 to 15·0) different programming languages. Intermediate subjects averaged 1·7 years (range = 1·0 to 3·0) experience and 2·7 (range = 2·0 to 5·0) programming languages. All of the advanced and intermediate subjects were familiar with one or more of the programming languages, Basic, Pascal, and FORTRAN. Other languages that had been used included APL, Cobol, Assembly, C, Prolog, and ALGOL. Novices averaged 0·25 years (range = 0·04 to 0·33) experience and an average of 1·4 (range = 1·0 to 3·0) languages. Novice subjects had used one or both of the languages, Pascal and Basic. Some novices had also used FORTRAN.

Materials

The set of 16 programming concepts presented in Table 1 served as stimuli. Originally, a larger set of 25 concepts was selected from chapter headings in Graham (1979), an introductory computer science textbook. Care was taken to select concepts that were not specific to a programming language, but that were basic to an

TABLE 1
Set of programming concepts

Repetition	Output	Subroutine	Function
Parameter	Sort	Search	Operator
Algorithm	Program	Assignment	Array
Global variable	Character data	Debug	Numerical data

understanding of programming. Results of pilot studies indicated that the subset of 16 items in Table 1 discriminated the most among programmers of varying degrees of experience. Although these 16 concepts clearly do not exhaust the domain of computer programming concepts, it was assumed that the set was sufficient to reveal some cognitive differences due to programming experience.

Procedure

Subjects were seated in front of a TERAK 8510 microcomputer upon which instructions were displayed. The instructions described a preliminary familiarity rating task in which subjects were to assign ratings on a scale from zero to nine to each of the 16 programming stimuli on the basis of their familiarity with the concept (zero = unfamiliar, nine = very familiar). The familiarity ratings were used mainly as a check to ensure that the concepts were familiar to at least the more experienced subjects, and that familiarity was not the sole basis for discrimination among groups.

After the familiarity ratings were completed, subjects were presented with additional instructions concerning a relatedness rating task. They were informed that there were generally several dimensions along which concepts could be related (e.g. similarity, frequency of co-occurrence, importance, generality). However, the instructions stressed that the subjects should base their rating on their first impression of relatedness. The scale ranged from zero to nine; zero indicated that the pair was highly unrelated and nine indicated that the pair was highly related. During the rating task, the scale with a movable bar marker was displayed along the top of the screen. The marker could be moved by pressing one of the keys marked zero through to nine on the keyboard.

After the instructions were read, subjects were presented with the complete list of 16 stimuli arranged in a random order so that they would be aware of the scope of the concepts that they would be rating. Following this presentation, the relatedness rating task began. All possible pairs of the 16 items (120 pairs) were presented one at a time and in a random order. The position of the items in each pair was counterbalanced across subjects. Each pair remained on the screen until the subject entered a rating. Subjects could change their rating during the presentation of a pair and indicated their final judgement by pressing the SPACE BAR which initiated the next trial.

RESULTS AND DISCUSSION

Relatedness ratings

In order to determine whether the general effect of experience on intragroup agreement was replicated in this study, relatedness ratings among subjects were correlated. The average Spearman correlations among individuals within the same group and in different groups are presented in Table 2.

Planned comparisons were performed to test for differences among these correlations using the Wilcoxin–Mann–Whitney test (Steele & Torrie, 1980). As anticipated, correlations among individuals within the same group were significantly greater than correlations among individuals in different groups (z ranged from 2·17 intermediate–advanced comparison to 10·73 for the naive–advanced comparison,

TABLE 2
Mean Spearman correlations of relatedness ratings among subjects

	Naive	Novice	Intermediate	Advanced
Naive	0·465†	0·187	0·178	0·119
Novice		0·207†	0·159	0·134
Intermediate			0·235†	0·268†
Advanced				0·387†

† $P < 0{\cdot}05$.

$P < 0{\cdot}05$ in all cases). Consistent with previous results (e.g. Adelson, 1981; McKeithen *et al.*, 1981; Soloway *et al.*, 1982), correlations among subjects in the advanced group were greater than those in the intermediate group ($z = 5{\cdot}54$, $P<0{\cdot}001$) and novice groups ($z = 6{\cdot}23$, $P < 0{\cdot}001$). However, the addition of the naive group resulted in a U-shaped relationship between experience and agreement. Naive subjects agreed slightly more than advanced subjects ($z = 2{\cdot}48$, $P < 0{\cdot}05$) and also more than novices ($z = 7{\cdot}09$, $P < 0{\cdot}001$) and intermediates ($z = 6{\cdot}55$, $P <$ 0·001). These results suggest that the naive programmers were using a common cognitive structure to assign the ratings. The low correlations between advanced programmers and naives, however, suggested that this naive structure was not the same structure that the advanced subjects used. Possibly, the naive structure was based on existing natural language relations for the concepts or common misconceptions about programming. The underlying basis for the naive agreement is further addressed in some analyses to follow.

In general, correlations indicated that the ratings of individuals in the four groups differed. It should be noted that mean familiarity ratings for the naive, novice, intermediate, and advanced groups were 4·9, 5·6, 7·6, and 7·5, respectively. Planned comparisons between pairs of groups revealed that familiarity ratings did not differ significantly between advanced programmers and intermediates or between naives and novices. Thus, it is unlikely that familiarity was the sole basis for differences in relatedness ratings.

Pathfinder networks

Relatedness ratings were converted to distances by subtraction from nine. The distances obtained from each subject were then submitted to the Pathfinder algorithm. The 40 sets of distance estimates that were submitted separately ultimately resulted in 40 different PFNETs. The Pathfinder algorithm can generate a family of PFNETs for each set of distance estimates. The PFNETs differ in density (i.e. number of links) depending on the values taken by parameters r and q. The r parameter is based on the Minkowski r-metric and defines the length of a path (one or more links) between two nodes. As r decreases, links are usually added to the PFNET. The parameter q defines the maximum number of links in a path and also affects PFNET density. Schvaneveldt *et al.* (1985*a*) provide additional details. The parameters $r =$ infinity and $q = n - 1$ (15 in this case) were chosen for the PFNETs constructed in this study. These parameters generate the simplest PFNET

(a tree unless there are ties) and require only ordinal assumptions to be made about the distance estimates.

The number of links in the 40 PFNETs ranged from 18 in an advanced PFNET to 58 in one of the naive PFNETs. It should be noted that each of the PFNETs contained one or more cycles (i.e. more than 15 links) and therefore could not be represented as a hierarchical tree structure. The average number of links was 36·9, 25·2, 25·7, and 25·1 for the naive, novice, intermediate, and advanced groups, respectively. These differences were significant, $F\ (3, 36) = 4{\cdot}46$, $MSe = 75{\cdot}21$, $P < 0{\cdot}01$. Paired comparisons indicated that this effect was due to the greater number of links present in the naive PFNETs than in the other PFNETs (Naive versus novice: $t(18) = 2{\cdot}76$, $SE = 4{\cdot}23$, $P < 0{\cdot}05$; naive versus intermediate: $t(18) = 2{\cdot}51$, $SE = 4{\cdot}46$, $P < 0{\cdot}05$; naive versus advanced: $t(18) = 2{\cdot}45$, $SE = 4{\cdot}81$, $P < 0{\cdot}05$). The large number of links in the naive PFNETs is indicative of a larger number of ties in the distance estimates. Examination of the frequency distributions of ratings indicated that the naive subjects tended to use the two end points of the rating scale more than other subjects. It is not surprising that naive programmers viewed the concept pairs as either related or unrelated. Lacking knowledge about the domain, they were unable to make any finer discriminations. Thus, this tendency to use a reduced scale resulted in more ties in the data and consequently more links in the PFNETs.

In order to simplify the examination of specific structural differences among the PFNETs of the four groups of subjects, distance estimates were averaged across subjects within each group and submitted to the Pathfinder algorithm. Averaging tends to reduce noise caused by spurious judgements and generates combined weightings in cases in which subjects judged relatedness along multiple dimensions. However, the averaging of distance estimates is problematic, particularly in cases in which intragroup correlations are low (e.g. the novice and intermediate groups). Therefore, the group PFNETs are supplemented with information about the individual PFNETs.

The four PFNETs that were generated using the average distance estimates are presented in Figs 1–4. The naive, novice, intermediate, and advanced PFNETs contained 19, 16, 15, and 16 links, respectively. Nodes were physically located on the page so as to minimize crossing of links. Each of the links is labelled with a link weight (indicating the strength or the weight of that link in the average structure). In addition, a "consensus" value is attached to each link. This value indicates the number of individual PFNETs out of the ten in the relevant group that contained that link. Thus, the consensus value provides a check on the averaging procedure. Links that have low consensus values should be interpreted with caution. On the other hand, every link in the four group PFNETs was present in at least two of the PFNETs for that particular group of subjects. Further, two-thirds or more of the links in each group PFNET were contained in the PFNETs of five or more individuals in that group.

Inspection of the group PFNET for the naive subjects revealed that many of the linked concept pairs had associations in natural language. For example, the concept *search* was linked to *sort, debug,* and *global variable.* Relations exist in natural language between *search* and each of these concepts (e.g., search and sort are two similar activities, eliminating bugs involves searching for them, a widespread search

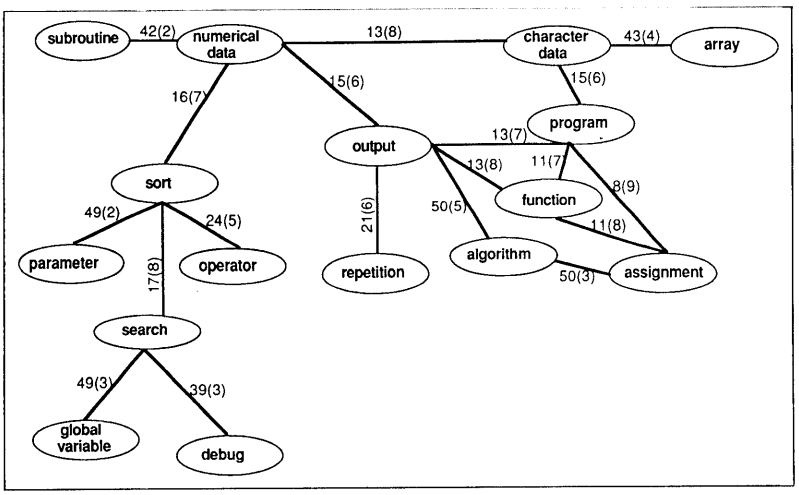

FIG. 1. Pathfinder network representation of the average naive cognitive structure. (Consensus values are in parentheses next to link weights.)

is a global search). As previously discussed, a possible explanation for the the relatively high naive intragroup correlations was that naive subjects based their ratings on a shared conceptual structure that had to do with the meanings of the terms in natural language. Thus, even though the subjects were told that the concepts were programming concepts, it is likely that they lacked a conceptual representation for many of the items in a programming context, and therefore,

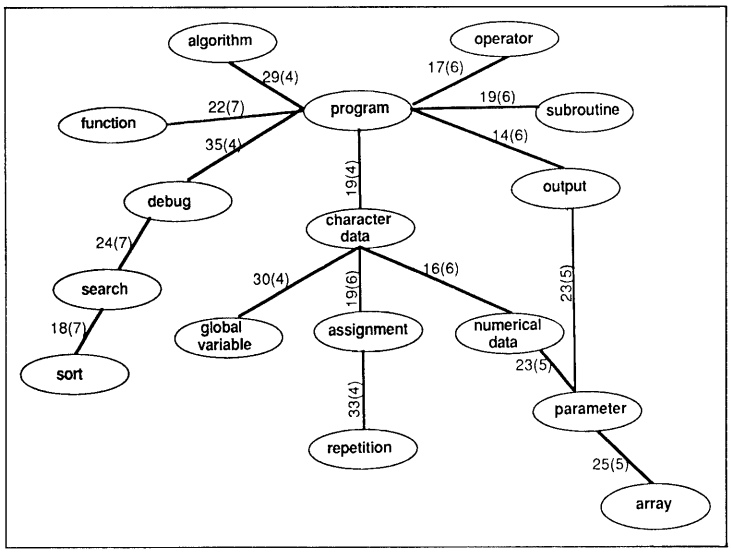

FIG. 2. Pathfinder network representation of the average novice cognitive structure. (Consensus values are in parentheses next to link weights.)

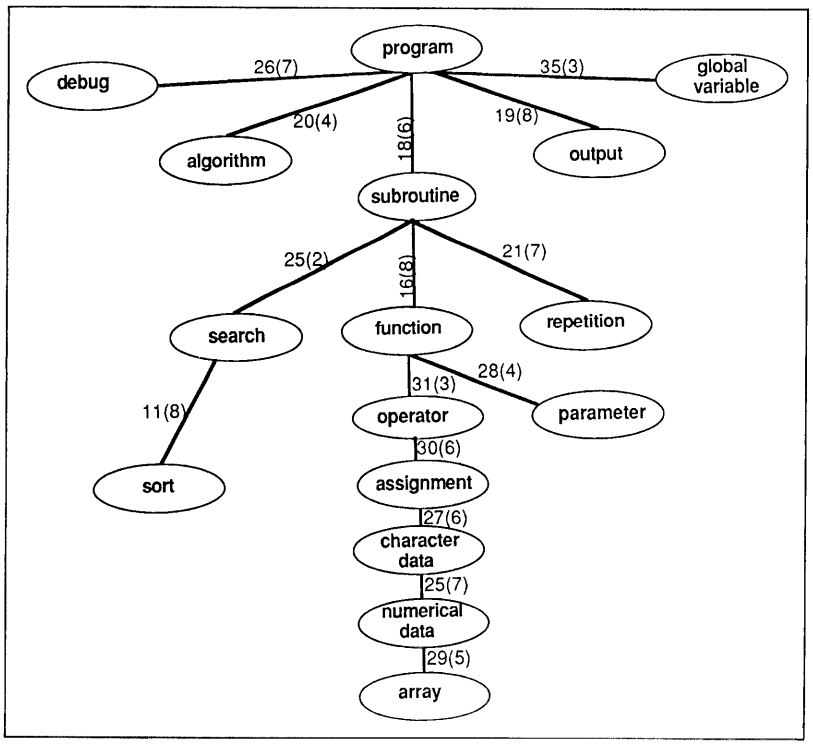

FIG. 3. Pathfinder network representation of the average intermediate cognitive structure. (Consensus values are in parentheses next to link weights.)

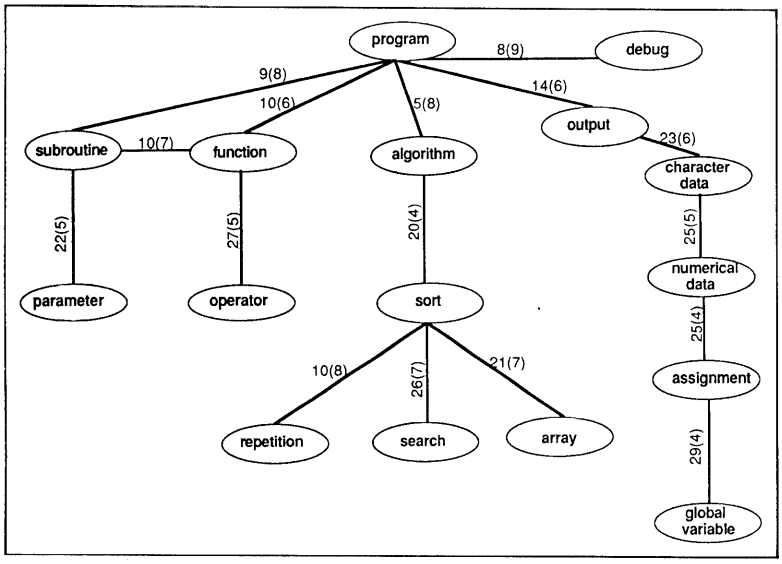

FIG. 4. Pathfinder network representation of the average advanced cognitive structure. (Consensus values are in parentheses next to link weights.)

made judgements on these items in terms of the associated concept outside of the programming domain. Whereas it is important to distinguish between concepts and words, it is also of interest to identify naive misconceptions that might be caused by associations of a programming concept to another for which prior knowledge exists.

It should be noted that the presence of a link in a PFNET implies that a relation exists; however, the Pathfinder output provides no explicit information concerning the nature of that relation (i.e. semantic link labels such as *is-a* or *has-a*). Consequently, although the links in the navie PFNET lend themselves to natural language associations, the present analysis provides no information relevant to this claim. Alternatively, naive subjects could have based their relatedness judgements on a naive model of computer programming which could have resulted in relations that were quite different from those based on the natural language meanings of the words. This issue is addressed in the second study in which subjects were asked to identify relations associated with links in the PFNETs.

The advanced PFNET revealed some interesting characteristics of programming experience. The cycle (*subroutine, function,* and *program*) present in the PFNET made sense at an intuitive level. Also, some concepts (e.g. *program, sort*) were connected to more concepts than others. These highly connected nodes might represent pivotal concepts in the memory organization of advanced programmers. Visual inspection of PFNETs is useful in the identification of specific structural features of each of the representations, but in order to summarize the similarities and differences of several PFNETs a more systematic, quantitative analysis is required. In the next section the PFNET structures are compared quantitatively in order to investigate structural differences across the four levels of experience.

Quantitative comparison of PFNETs

An examination of Figs 1–4 suggests that the four group PFNETs differ, although the extent of the differences is difficult to determine by visual inspection. One would expect, for instance, the advanced structure to be more like the intermediate structure than the naive structure because the former group is more similar in terms of programming experience. However, such information is not immediately apparent in the PFNETs. Therefore, in order to determine how the group PFNETs resemble each other, correlations were performed on the structures.

Each of the four group PFNETs was represented as a vector of 120 values (one for each pair of concepts). The values were the sum of the link weights along the shortest path in the PFNET connecting each pair of nodes. Consequently, for linked pairs the value was simply the link weight. Spearman correlations were performed on pairs of these vectors and are presented in Table 3. In general, the correlations were higher for groups that were adjacent to each other, such as advanced and intermediates (mean $r = 0{\cdot}261$), than for groups that were separated by one (mean $r = 0{\cdot}176$) or by two (mean $r = 0{\cdot}063$) other groups. These results lend support to the psychological validity of the Pathfinder representations in that, as one would expect, the PFNETs of groups with similar levels of experience were more alike than those of groups with different amounts of experience.

Results thus far have indicated that the subjects in the different experience groups rated pairs of programming concepts differently and that the PFNETs that were generated from the ratings were also structurally different. Furthermore, the

TABLE 3
Spearman correlations of group PFNETs

	Naive	Novice	Intermediate	Advanced
Naive	1·000	0·150	0·076	0·063
Novice		1·000	0·213†	0·279†
Intermediate			1·000	0·419†
Advanced				1·000

† $P < 0{\cdot}05$.

differences between group PFNETs tended to be smaller for groups of subjects with similar amounts of programming experience. However, these analyses only quantified the overall similarities (or differences) of the PFNETs. It is also important to identify specific features of the representations that underly the overall similarities and differences. For instance, what features of the intermediate representation make it more like the advanced representation than the naive representation? One specific way that the group PFNETs differ is in the way that specific concepts are represented in each structure. In this paper the representation of a particular concept is defined as consisting of the set of all other concepts to which it is directly linked in the PFNET. For any two PFNETs, the representation for some concepts may be quite similar, whereas for other concepts they may differ. For example, *function* is linked to the concepts *output, program,* and *assignment* in the naive PFNET, to *program* in the novice PFNET, to *parameter, operator,* and *subroutine* in the intermediate PFNET, and to *program, operator,* and *subroutine* in the advanced PFNET.

Such differences in conceptual representation were systematically analysed using a strategy similar to one employed by Schvaneveldt *et al.* (1985*b*) in order to identify concepts that were understood (or misunderstood) by undergraduate pilot trainees. Degree of concept understanding by undergraduate pilots was defined in terms of a comparison of the undergraduate conceptual representation to the representation of the same concept in the expert (instructor pilot) PFNET. A concept in an undergraduate PFNET could be linked to the same concepts as in the instructor PFNET, indicating an understanding of the concept at the instructor level, or linked to different concepts indicating a lack of advanced understanding of that particular concept. Thus, in the following analyses, advanced conceptual representations were used as standards against which to evaluate concept understanding of the less experienced programmers.

Before conceptual differences in PFNETs were analysed, it was important to demonstrate that they existed. That is, were individual concepts represented differently (i.e. linked to different sets of concepts) in the PFNETs of the advanced programmers from what they were in the intermediate PFNETs? This idea was quantified by representing each concept (the target concept) for each of the four groups as a vector of 15 values corresponding to the 15 other concepts. Each value took the number of individual PFNETs in that group in which a link occurred between that concept and the target concept. Thus, values ranged from 0 in cases in which the concept and the target were not linked in any of the PFNETs to 10 in

cases in which they were linked in all of the PFNETs for that group. For each concept, a difference value for any two groups can be computed by summing the absolute differences of the two vectors for those groups. The larger the difference value, the more disparate the two conceptual representations.

Difference values were computed for each concept and for each of three comparisons, advanced versus naive, advanced versus novice, and advanced versus intermediate. Mean differences were 35·50, 27·28, and 20·00 for the naive, novice, and intermediate comparisons, respectively. These differences were significant, $F(2, 30) = 22{\cdot}084$, $MSe = 43{\cdot}55$, $P < 0{\cdot}001$. All three paired comparisons were also significant: naive/advanced versus novice/advanced, $t(30) = 2{\cdot}756$, $SE = 2{\cdot}948$, $P < 0{\cdot}01$; naive/advanced versus intermediate/advanced $t(30) = 5{\cdot}268$, $SE = 2{\cdot}943$, $P < 0{\cdot}001$; and novice/advanced versus intermediate/advanced $t(30) = 4{\cdot}178$, $SE = 1{\cdot}765$, $P < 0{\cdot}001$. The effect of concepts, however, was not significant. These results indicated that there were differences in the way that concepts were represented in the advanced PFNETs and the PFNETs of less experience programmers and that these differences decreased with increasing experience.

The above results indicated that, in general, naive, novice, and intermediate PFNETs have conceptual representations that differ from the advanced conceptual representations. Next, a fine grained analysis was performed to discriminate between concepts on the basis of degree of understanding. In order to identify concepts that were poorly understood, a median split was performed on the difference values for the 16 concepts and three comparisons. Concepts that were associated with low difference values based on this split were classified as well-defined and those that had high difference values were classified as misdefined. The resulting classifications of concepts for naive, novice, and intermediate groups is presented in Table 4.

TABLE 4

Misdefined concepts based on comparison of representations of concepts to advanced representations of concepts

Intermediate	Novice	Naive
repetition	repetition	repetition
sort	sort	sort
array	array	
	function	function
	search	search
	algorithm	algorithm
program	program	
	assignment	assignment
	parameter	
	debug	
		output
		subroutine
		operator
		character data
		numerical data

The number of misdefined concepts decreased with increasing expertise. However, the differences were only marginally significant, $\chi^2(2) = 5{\cdot}362$, $P < 0{\cdot}10$. Also, most of the concepts progressed systematically from misdefined to well-defined. For instance, the concept *function* was misdefined at the naive and novice level, but became well-defined at the intermediate level. However, there were exceptions to this general rule. Specifically, the concepts *array, program, parameter,* and *debug* were well-defined at the naive level and became misdefined at the novice level. There are several plausible explanations for this result, including noise and lack of sensitivity of the techniques. Another possible explanation for this "regression in learning" is that the naive programmers coincidentally had the correct links for these concepts because their understanding of them (in natural language, for instance) happened to involve the advanced links. In other words, all links are not created equal. Two individuals may both view a pair of concepts as related, but along completely different dimensions. Again, a detailed examination of link differences requires the presence of semantic relations. These issues will be addressed further in the second study. Of course, it is also possible, although unfortunately so, that the regression in concept understanding was real in that concepts that were once clearly understood became confused and unclear with learning.

In summary, one way to view the overall differences among the four representations is in terms of differences in the way that individual concepts are represented. For instance, the advanced PFNET is more like the intermediate PFNET than the novice or naive PFNETs, and this similarity can be attributed to the fact that only four of the 16 concepts that are in the intermediate PFNET are represented differently in the advanced PFNET (misdefined), whereas 10 and 11 are misdefined in the novice and naive PFNETs, respectively. Thus, the conceptual analysis provides a means of specifying overall differences in terms of particular concepts. In addition, the finding that the intermediate programmers had very few misdefined concepts and that the majority of the concepts progressed with experience from misdefined to well-defined lends support to the psychological validity of the PFNET representations.

Study 2

The analyses in the previous study were limited in the sense that only presence or absence of links was considered and not the meaning of the links (i.e. specific type of semantic relation). The fact that a concept pair is linked in two different PFNETs merely indicates that the two groups of subjects both viewed this pair as related; however, the two groups could differ on the specific type of relation. Therefore, the purpose of this study was to attempt to identify the meaning of the links in the four PFNETs.

METHOD

Subjects

Twenty subjects (five in each of the four groups defined in Study 1) participated in this study. Subjects consisted of introductory psychology students at New Mexico

State University, who participated in order to fulfill a research requirement, and volunteer graduate students from the computer science department at New Mexico State University.

Materials

Materials consisted of the four sets of linked concept pairs from each of the four group PFNETs generated in Study 1. Individual PFNETs were not used in this study because it did not seem appropriate to ask subjects to label the linked pairs that another individual viewed as related. There were 19, 16, 15, and 16 pairs of concepts in the set for the naive, novice, intermediate, and advanced groups, respectively.

Procedure

Each subject was presented with the set of linked concept pairs derived from the PFNET corresponding to that subject's level of programming experience. Pairs were listed (one per row) on a sheet of paper. The order of the pairs and the position of each concept within a pair were randomized over subjects within each skill group. In the first phase of the study subjects were told to write beside each pair a sentence or phrase in the form "concept–relation–concept". They were also told that they could reverse the order of the two concepts if they desired. In the second part of the study subjects were presented with a second copy of the same list of pairs and performed the same task except that they were to select the relation for each pair from a list of 29 relations. The list of relations was the same for all subjects and included such phrases as "is part of", "occurs with", "precedes", and "is done to". The list of relations was based on the results of a pilot study that was identical to the first task in this study.

RESULTS AND DISCUSSION

The semantic relation associated with a link was considered to be identified if three or more out of the five subjects in each group agreed on a label in at least one of the two tasks. In the multiple choice task, two relations were judged as the same only if they were identical. In the sentence formation task, two relations were judged as the same if they were identical except for the following conditions: (1) reversals which preserved meaning (e.g. a function *is part of* a program, a program *contains* a function); (2) additions or deletions of words not affecting the meaning (e.g. search *for an* array, search *can be for* an array); and (3) the use of synonyms (e.g. *is a type of, is a kind of*). In cases in which there was more than one version of the same relation, the version that was used by the majority of the subjects was selected as the semantic relation for that link. For each pair in which three or more subjects agreed on a relation, the semantic relation as well as the direction of the relation were noted.

A total of 13 of the 16 advanced links (81%) were identified (according to the above criteria) by the five advanced subjects in this study. Five of the links were identified in the sentence formation task, five in the multiple choice task, and three in both tasks. For the intermediate group, nine of the 15 links were identified (60%) and of these, five were identified in the sentence formation task, three in the multiple choice task, and one in both. Of the five (31%) of the novice links

TABLE 5
Semantic labels for linked concept pairs that were identified by one or more groups

Linked pair	Semantic relation			
	Advanced	Intermediate	Novice	Naive
subroutine–program	is part of	is part of	is part of	XXX
character data–output	is a type of	XXX	XXX	XXX
parameter–subroutine	is used with	XXX	XXX	XXX
program–output	produces	produces	—	—
sort–search	involves	—	—	is before
function–operator	is a	—	XXX	XXX
function–program	is a part of	XXX	—	—
debug–program	is done to	is done to	is done to	XXX
function–subroutine	is a	⟨is a⟩	XXX	XXX
repetition–sort	is part of	XXX	XXX	XXX
program–algorithm	is the implementation of	⟨is part of⟩	⟨is part of⟩	XXX
sort–algorithm	is a	XXX	XXX	XXX
sort–array	is done to	XXX	XXX	XXX
array–numerical data	XXX	can consist of	XXX	XXX
operator–assignment	XXX	does	XXX	XXX
search–subroutine	XXX	for a	XXX	XXX
program–global variable	XXX	contains	XXX	XXX
character data–assignment	XXX	—	is part of	XXX
assignment–repetition	XXX	XXX	can be	XXX
sort–operator	XXX	XXX	XXX	is done by
sort–numerical data	XXX	XXX	XXX	is done to
search–global variable	XXX	XXX	XXX	is done for a

Note: XXX indicates that the link is not present in the PFNET for that group; — indicates that the link is present in the PFNET for that group, but the label was not identified; ⟨⟩ indicates that the position of concepts is reversed for this relation.

identified, three were identified in the sentence completion task and two in the multiple choice task. Finally, each of the four (21%) identified naive links was identified in the sentence formation task. The semantic relations that were identified for each of the four PFNETs are presented in Table 5.

Consistent with previous findings on agreement and expertise, agreement on labels among subjects increased with increasing experience. The absence of the U-shaped agreement function seen previously suggests that although naive subjects agreed that certain pairs were more related than others, they did not agree on the specific nature of the relation. This could be due to the fact that the naive programmers know very little about these concepts, so that although their hunches or guesses agreed with each other, they were unable to identify and verbally report a relation. On the other hand, these concepts could be related in several ways (particularly outside of the programming domain) and naive subjects might have differed on the specific relation that they viewed as salient for a pair of concepts.

Throughout this paper it has been suggested that many of the naive misconceptions about programming might originate from pre-existing "naive" knowledge about the programming concepts or pre-existing knowledge about natural language

concepts corresponding to the programming terms. The labels that were generated by the naive subjects supported this latter view. For example, naive programmers labelled the *sort–search* link with the relation *is before*, whereas advanced programmers assigned the relation *involves* to this link. In everyday usage of these terms, sorting could be considered an independent process that typically precedes searching; however, in terms of computer programming a searching procedure is usually called within a sort program. In this example it appears that the relational discrepancy was due to subtle differences between the meaning of the concepts in the programming domain and their meaning in everyday natural language. Similarly, the other naive links that were identified could also be interpreted as natural language relations.

It is interesting to compare the discrepancies and consistencies present in the link labels of different groups. There were 22 links that were labelled by at least one group, and of these there were nine links that occurred in more than one of the PFNETs. Thus, of all the labelled links, there were nine links that two or more groups were asked to label. In three of these nine cases, the link label was identified by only one group, whereas the other group or groups were unable to agree on a label. Of the remaining six links, there were three for which the labels were the same across groups and three for which the labels were different across groups. The concept pair *program–algorithm* is an example of a linked pair that was labelled differently by different groups. The advanced programmers' label for this pair suggested that a program is an implementation of an algorithm, whereas the label that novices and intermediates assigned to this pair indicated that an algorithm is part of a program. Such relational discrepancies provide clues concerning possible misconceptions of less experienced programmers.

In the previous analysis of concept understanding, the meaning of the link was not considered. However, it is now possible to compare the labels of the links that the less experienced programmers shared with the advanced programmers. The greater the proportion of conflicting labels among the shared, labelled links, the more likely it would be that some concepts that were classified as well-defined in the first analysis were actually misdefined.

Of the four links that were labelled in the naive PFNET, only one of them (*sort–search*) was also in the advanced PFNET, and, as mentioned previously, the naive label conflicted with the advanced label. The novice PFNET shared three of the five labelled links with the advanced PFNET, and one of them (*program–algorithm*) was labelled differently by the two groups. However, in both the naive and novice cases the concepts that make up the conflicting pair were already classified as misdefined, so the results would not be altered by this difference. Five of the nine labelled intermediate links were also contained in the advanced PFNET, and two of these (*function–subroutine, program–algorithm*) were assigned labels that conflicted with the advanced label. Each concept in these two pairs was classified as well-defined in the concept understanding analysis. Thus, the concept understanding analysis was performed again, ignoring these two conflicting links in the intermediate network. Results indicated that out of the four concepts, the concepts *function* and *algorithm* would be classified as misdefined because of the conflicting link labels. In general, there were very few conflicting link labels and the ones that did exist has little effect on the previous analysis.

The semantic relations also augmented the analysis of structural similarity between groups with similar amounts of experience. As mentioned above, there were three links for which the labels given by two or more groups were identical. In all of these cases the groups that agreed on the link label were next to each other in terms of the four point experience scale used in this study. That is, it was never the case that the novices and advanced programmers agreed with each other, but not with the intermediates on a link label. Alternatively, if the advanced programmers agreed with any group on a link label it was the intermediates. This type of systematic consistency among groups not only supports the structural similarities seen earlier but it also increases confidence in these paricular link labels.

Finally, it was quite difficult to identify any semantic relation for many of the links, especially for the less experienced subjects. Not only was intersubject agreement on a particular label a problem, but in several cases individuals had trouble attaching any label to a link (e.g. *numerical data–character data*). Anderson (1983) has suggested that general relatedness decisions are much easier to make than decisions concerning the specific nature of the relation. It could be that this type of information is automatic (Shiffrin & Schneider, 1977) or compiled (Anderson, 1982) and, therefore, unavailable to conscious awareness. In fact, research by Nisbett and Wilson (1977) has suggested that verbal reports are often incomplete and inaccurate. Consequently, more work is needed on the development of methodologies which facilitate this link labelling process.

General discussion

In this study, programming knowledge was represented in empirically derived networks for naive, novice, intermediate, and advanced programmers. The PFNETs not only made sense at an intuitive level, but quantitative information extracted from the PFNETs supported the psychological meaningfulness of this form of representation. Correlations of the PFNET structures indicated that the PFNETs varied systematically with expertise, with correlations being highest for PFNETs of those groups closest in level of experience. In addition, concept understanding, as defined by the set of other concepts linked to the concept in question, increased with experience. Finally, the identification of semantic relations corresponding to links in the PFNETs added some necessary information to the structures.

It should be emphasized that the PFNETs are representations of cognitive structure. Without additional control processes or inference rules, the PFNETs cannot behave intelligently (e.g. answer questions or make decisions). Research is being conducted to investigate various strategies for implementing activation processes in PFNETs (Schvaneveldt, 1987). However, the purpose of this research was to investigate differences in cognitive structures. Thus, implicit in this research is the assumption that cognitive structure can be studied apart from the control processes that operate on it.

Secondly, it is important to point out that these studies dealt with knowledge about abstract programming concepts. Even with complex inference rules, this structure alone would not be sufficient to implement a system that wrote computer programs. However, it is clear that programmers must need to know what a global variable is and what a function is in order to do programming. The structures in this

paper could be viewed as representations of declarative knowledge (Anderson, 1982). With a suitable set of inference rules, these structures could be used to answer questions about the meanings of programming terms. On the other hand, whereas the PFNETs in these studies were derived from relatedness ratings, they could also be derived from protocols taken from programmers as they actually write code. Programming steps or actions could be represented as nodes and distance estimates could be derived from the distance between two actions in the protocol. The resulting structure would represent procedural knowledge needed to actually write programs. It seems likely that knowledge about writing programs interacts with knowledge about the meaning of programming concepts.

The results of this research revealed some interesting characteristics of programmer knowledge. In particular, the relatedness rating correlations indicated that intragroup agreement did not increase linearly with expertise, but instead decreased initially from the naive to novice level and then gradually increased from the novice to advanced level. It is interesting to speculate on the meaning of this U-shaped function. Generally, the higher correlations indicated that individuals within each group (i.e. naive and advanced programmers) shared a common cognitive structure, although the structures were not the same for the two groups.

The naive structure appeared to be based on the meaning of the terms in natural language, whereas the advanced organization was apparently based on the meaning of the concepts in the programming domain. Although the links that were successfully identified supported these hypotheses, there were many links (especially in the naive PFNET) that subjects were unable to label. In order to interpret the various representations adequately, this type of information is needed. Recently, techniques have been developed that aid in indirectly eliciting information about links in PFNETs from subjects (Cooke, 1987). The application of such techniques to the data in these studies might add information to the PFNETs that would enable stronger claims to be made concerning the basis for the representations.

The fact that naive programmers shared some programming knowledge, whether right or wrong, should be of interest to those who teach programming. The naive cognitive structure could be thought of as a naive mental model of programming, just as the "person on the street' has a naive mental model of physics (McCloskey, 1983). Misconceptions in the model could be immediately corrected and appropriate conceptualizations encouraged. The concept understanding analysis in Study 1 provides one means of identifying possible misconceptions in naive mental models. Additional work is needed to demonstrate that the misconceptions that are predicted by conceptual representations in PFNETs are indeed concepts with which beginning programmers have difficulty. For instance, a vocabulary test could be administered to students taking their first programming course at various points throughout the semester to determine whether the concepts classified as misconceptions on the basis of the PFNETs corresponded to the concepts that were missed the most on the test.

Interestingly, subjects who were exposed to some programming experience no longer shared a cognitive structure. This decline in agreement could be due to variations in teaching strategies, text books, teachers, or programming languages learned at the novice and intermediate levels. At some point (i.e. advanced), these basic concepts became well-learned, so that these variations in type of programming

experience no longer mattered. Thus, the decline in intragroup agreement does not necessarily imply a "confused cognitive structure", but rather the existence of several different structures within the group. Variations in cognitive structure with different types of experience in the same domain is an issue for further research.

This research has specific applications to programmer education or training. The advanced PFNET representation can be considered an explicit goal state in the learning process. The naive representations can provide clues to the state of naive knowledge prior to learning, and can serve as a means of periodically evaluating the cognitive development of students. In teaching situations it may be useful to draw attention to the difference between the experts' and nonexperts' views of particular relations. For example, based on this work, one might point out the difference between a program being the implementation of an algorithm and an algorithm being part of a program. Furthermore, various teaching or training strategies could be evaluated by comparing students' progress in concept understanding for each strategy.

The PFNET representations of cognitive structure can also be useful in the design of expert systems. The acquisition of knowledge from a human expert is one of the most time-consuming phases of expert system development. Furthermore, humans, especially experts in a domain, often have difficulty introspecting about their knowledge and expressing their knowledge verbally. Pathfinder networks could provide a representation of tacit expert knowledge as well as a means of organizing facts and rules in the knowledge base. Thus, the advanced programmer PFNET might provide useful information to those developing expert systems in the programming domain.

The authors would like to thank the faculty and students of the psychology and computer science departments who participated in this study. Thanks also to Karen Preuss and Mel Tempel who assisted in the data collection for the first study.

References

ADELSON, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition,* **9,** 422–433.

ADELSON, B. (1984). When novices surpass experts: the difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition,* **10,** 483–495.

ADELSON, B. (1985). Comparing natural and abstract categories: a case study from computer science. *Cognitive Science,* **9,** 417–430.

ANDERSON, J. R. (1982). Acquisition of cognitive skill. *Psychological Review,* **89,** 369–406.

ANDERSON, J. R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

CHASE, W. G. & SIMON, H. A. (1973). Perception in chess. *Cognitive Psychology,* **5,** 121–152.

CHI, M. T. H., FELTOVICH, P. J. & GLASER, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science,* **5,** 121–152.

COOKE, N. M. (1985). Modelling human expertise in expert systems. *Memorandum in Computer and Cognitive Science,* MCCS-85-12, Computing Research Laboratory, New Mexico State University.

COOKE, N. M. (1987). *The Elicitation of Units of Knowledge and Relations: Enhancing Empirically Derived Semantic Networks.* Unpublished PhD thesis, New Mexico State University.

COOKE, N. M., DURSO, F. T. & SCHVANEVELDT, R. W. (1986). Recall and measures of

memory organization. *Journal of Experimental Psychology: Learning, Memory, and Cognition,* **12,** 538–549.

COOKE, N. M. & MCDONALD, J. E. (1986). A formal methodology for acquiring and representing expert knowledge. *Proceedings of the IEEE: Special Issue on Knowledge Representation,* **4,** 1422–1430.

COOKE, N. M. & MCDONALD, J. E. (1987). The application of psychological scaling techniques to knowledge elicitation for knowledge-based systems. *International Journal of Man–Machine Studies,* **26,** 533–550.

DEARHOLT, D. W., SCHVANEVELDT, R. W. & DURSO, F. T. (1985). Properties of networks derived from proximities. *Memorandum in Computer and Cognitive Science,* MCCS-85-14, Computing Research Laboratory, New Mexico State University.

DREYFUS, S. E. & DREYFUS, H. L. (1987). Towards a reconciliation of phenomenology and AI. In D. PARTRIDGE, & Y. WILKS, Eds, *A Source Book on the Foundations of AI.* London, UK: Cambridge University Press.

ENGLE, R. W. & BUKSTEL, L. (1978). Memory processes among bridge players of differing expertise. *American Journal of Psychology,* **91,** 673–689.

GRAHAM, N. (1979). *Introduction to Computer Science: A Structured Approach.* St Paul, MN: West Publishing Co.

MCCLOSKEY, M. (1983). Naive theories of motion. In D. GENTNER, & A. STEVENS, Eds, *Mental Models.* Hillsdale, NJ: Erlbaum.

MCDONALD, J. E., DEARHOLT, D. W., PAAP, K. & SCHVANEVELDT, R. W. (1986). A formal interface design methodology based on user knowledge. *Proceedings of Human Factors in Computer Systems, CHI'86,* pp. 285–290.

MCKEITHEN, K. B., REITMAN, J. S., RUETER, H. H. & HIRTLE, S. C. (1981). Knowledge organization and skill differences in computer programmers. *Cognitive Psychology,* **13,** 307–325.

MURPHY, G. L. & WRIGHT, J. C. (1984). Changes in conceptual structure with expertise: differences between real-world experts and novices. *Journal of Experimental Psychology, Learning, Memory, and Cognition,* **10,** 144–155.

NISBETT, R. E. & WILSON, T. D. (1977). Telling more than we can know: verbal reports on metal processes. *Psychological Review,* **8,** 231–259.

REITMAN, J. S. (1976). Skilled perception in Go: deducing memory structure from inter-response times. *Cognitive Psychology,* **8,** 336–356.

ROSKE-HOFSTRAND, R. J. & PAAP, K. R. (1986). Cognitive networks as a guide to menu organization: an application in the automated cockpit. *Ergonomics,* **29,** 1301–1312.

SCHOENFELD, A. & HERRMANN, D. (1982). Problem perception and knowledge structures in expert and novice mathematical problem solvers. *Journal of Experimental Psychology: Learning, Memory, and Cognition,* **8,** 484–494.

SCHVANEVELDT, R. W. (1987). Proximities and schemata. *Invited paper on a Symposium at the meetings of the Rocky Mountain Psychological Association, Albuquerque, NM.*

SCHVANEVELDT, R. W., DURSO, F. T. & DEARHOLT, D. W. (1985*a*). Pathfinder: scaling with network structures. *Memorandum in Computer and Cognitive Science,* MCCS-85-9, Computing Research Laboratory, New Mexico State University.

SCHVANEVELDT, R. W., DURSO, F. T., GOLDSMITH, T. E., BREEN, T. B., COOKE, N. M., TUCKER, R. G. & DEMAIO, J. C. (1985*b*). Measuring the structure of expertise. *International Journal of Man–Machine Studies,* **23,** 699–728.

SHIFFRIN, R. M. & SCHNEIDER, W. (1977). Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. *Psychological Review,* **84,** 127–190.

SHNEIDERMAN, B. (1976). Exploratory experiments in programmer behavior. *International Journal of Computer and Information Sciences,* **5,** 123–143.

SHNEIDERMAN, B. (1980). *Software Psychology: Human Factors in Computer and Information Systems.* Boston, MA: Little, Brown.

SOLOWAY, E., EHRLICH, K. & BONAR, J. (1982). Tapping into TACIT programming knowledge. *Proceedings of Human Factors in Computer Systems,* 52–57.

STEEL, R. G. & TORRIE, J. H. (1980). *Principles and Procedures of Statistics.* New York: McGraw-Hill.