CONCEPTUAL GRAPH KNOWLEDGE SYSTEMS AS PROBLEM CONTEXT
FOR NEURAL NETWORKS

George G. Lendaris
Systems Science Ph.D. Program
Portland State University
Portland, OR 97207-0751

ABSTRACT

The key motivating objective in connectionist network research is to achieve nets (architectures and training strategies) that will be able to solve problems in "real-world" contexts. Associated with any problem-solving context is constraint (patterns). For a net to be able to learn and to generalize well, there must be some correspondence between the structure/constraints of the net's architecture and those of the given problem space. It is fundamental, therefore, that eventually, recourse to experiments with "real-world" problems will always be required in connectionist research. This paper gives an outline of a problem area for which connectionist nets could hold great promise: knowledge systems, where the knowledge is encoded/represented via conceptual graphs. Certain aspects of this problem context are already known, and these are probed for possible implementation via connectionist nets.

1. INTRODUCTION

In connectionist research, theoretical and application activities must be close companions—each to guide the other. This is due to an implicit requirement that constraints in the problem domain be somehow mirrored in the architecture of the neural network being trained in that problem domain for successful learning to even be possible [Lendaris & Stanley, 1965]. This requirement derives from the following facts: a connectionist network begins its learning process with a particular set of elements, a particular pattern of connections among them, and a particular set of connection weight values; this architecture determines precisely the set of input–output mappings (functions) achievable by that net as the weights are adjusted; the achievable functions are, typically, a subset of all the possible functions (i.e., a constrained set); and, for the net to learn a specific task, the mapping corresponding to the task must be contained within the constrained set of mappings achievable by that net. [If the connections and/or the transfer functions of elements are changed during training, the constraint on achievable functions may be relaxed some, but the principle is the same.]

In this paper, a "real-world" problem domain is suggested which could benefit greatly from the order(s) of magnitude speed up in processing available via neural networks (plus benefit from other attributes of neural nets: fault tolerance, associative recall, etc.), and at the same time provide connectionist researchers with a context from which to guide their research. The problem domain is that of conceptual graph knowledge systems.

The approach here is to present some basic properties of conceptual graphs, indicate operations important in their application, and point out those that might be candidates for implementation with neural nets. A special representation schema for conceptual graphs is required for their implementation via neural nets. The representation used here is developed in [Lendaris, 1988].

Presented at International Conference on Neural Networks (ICNN-88)
San Diego, CA, July, 1988
2. CONCEPTUAL GRAPH BASICS

Conceptual graphs are defined in detail in [Sowa, 1984] and in [Sowa, et al, 1988]. Of specific interest here is the fact that a conceptual graph consists of two different kinds of nodes (concept nodes and relation nodes) and directed arcs which serve to show connections among the nodes. An arc connects a concept node to a relation node or a relation node to a concept node; connections are not allowed between nodes of the same type. An example of a generic conceptual graph is given in Figure 1.

Circles with relation labels, squares with concept labels, and lines with arrows on them are used to depict a conceptual graph in a graphical format—called the "display form," usually the form preferred by human observers. An alternative "linear form" with parentheses around relation labels, brackets around concept labels, and arrow indicators is used to represent a conceptual graph for easier input to computers via keyboard, and output via conventional printers. The linear form is also analogous to computer data organizations which use pointers for linking data items. This latter form has been useful for the serial-type, symbolic processing typical in implementations to date. See Figure 2 for a linear-form representation of the conceptual graph shown via the display form in Figure 1.

There are four "canonical" operations on conceptual graphs (JOIN, SIMPLIFY, COPY, RESTRICT) which are used to effect most other graph operations; an example of the latter is one called PROJECTION. This operation is the basis of the application discussed later in the paper.

Knowledge systems using Sowa's formalism typically comprise the following components:

1. First, a knowledge engineer gathers together data which are coalesced into (true) assertions, and these in turn are represented via conceptual graphs (CGs). We assume here that a complete catalog of the conceptual graphs developed by the knowledge engineer(s) is available, say, in a linear form. [Note: The entire problem context (typically represented via a semantic network, which may or may not use the Sowa conceptual-graph formalism) is taken into account when defining and choosing the concept and relation labels to be used in the CGs.]

2. A catalog of all the concept types is created and maintained.

3. A hierarchy tree is created and maintained which encodes the sub- and super-type relationships among the concept types. This hierarchy is used for checking validity of restrictions and generalizations of CGs.

4. A catalog of all the relation types is created and maintained.

4a. A relation definition dictionary is created and maintained. For each type in the relation catalog, there is listed a set of rules (constraints) which dictate the number of arcs leaving and entering the node, and the maximal type of the concept to which each arc may be attached (maximal here means the highest concept type in the hierarchy to which it is allowed to connect the given arc).

5. A restriction operates on a concept to yield either a concept which is a sub-type of the original [e.g., person to man], or, a concept which has been specialized to an individual in the context [e.g., person to person:Gregory]—also called an instantiation.

6. A generalization operates on a concept to yield a concept which is a super-type of the original [e.g., man to person].

7. A catalog of names for individuals (also called markers) appearing in the knowledge base is created and maintained. Each marker in this list carries an indication of its associated concept sub-type; the latter is used for making conformity checks.

8. The conformity operation checks to determine whether the concept type associated with the name of the individual assigned during a restriction is appropriate to the concept to which it was assigned—e.g., man:Gregory vs. woman:Gregory.
The test for PROPERTY 1 (concurrence of relation types) is particularly simple, and is described in [Lendaris, 1988] using standard parallel registers. Implementation using neural nets should be straightforward.

The second, or PROPERTY 2, test is a bit more complicated than the first, and is the one that can particularly benefit from neural net type processing. This test is the subject of the remainder of this paper.

The added difficulty of this test is occasioned by the fact that a SUBtype of the concepts is allowed (the RESTRICT operation is used here). This entails matching each concept type represented in the Templ-C-Vector against the concept types found in the Fact-C-Vector of a candidate FactCG, and demonstrating that for each concept type in the query concept, a subtype is found in the candidate FactCG. To check for the existence of a subtype relationship between a pair of concept types, the concept-type hierarchy will have to be accessed.

4. USING NEURAL NETWORK for TEST of PROJECTION PROPERTY 2

The best situation would be if a connectionist network could perform the entire PROPERTY-2 test in one step. A procedure is described for using a connectionist network with this capability assumed. In addition, however, two other procedures are given, each using a connectionist network of successively less capability. For even the lowest assumed capability, it is argued that use of connectionist networks to perform the suggested operations on conceptual graphs can still effect a significant reduction of processing time.

In a recent paper, Hinton [1986] reports results of an experiment wherein a connectionist network with a particular kind of structure (three hidden layers) was used to learn the family relationships between 24 different persons. As is well known, family relationships are representable as a hierarchy, much as the subtype relationships needed in the present context. In fact, the concept-type hierarchy for the present context appears to be less complex than the family relationships handled by Hinton, because in the family case, two trees were merged to represent a marriage of individuals from the two trees. In Hinton's example, a separate input to the connection network was provided for each individual family member. A separate input for each of the relationships was also provided [there were 12 of these: husband, wife, son, daughter, brother, sister, father, mother, aunt, uncle, niece, nephew]. For the data-base application suggested in the previous Section, the number of relationships is much smaller, but the number of concept types is larger than the 24 persons Hinton used. Nevertheless, taking Hinton's results as a demonstration that connectionist networks can be successfully taught to learn hierarchical type relationships, we propose here the following kind of connectionist network "black box."

The connectionist network box is to have 2 groups of numC (the number of concept types in the knowledge base catalog) input terminals. Each of the terminals in the first group will be assigned to one of the concept types, and a similar assignment given to the terminals in the second group. The assignment of concept types to these terminals is made to correspond in a useful way to the assignment of concept types to the slots in the C-vector template. Having two groups of input terminals allows presenting concepts from two conceptual graphs as ordered pairs, and asking about their relative level on the hierarchy. The output of the network may be as simple as four wires; one each for the answers: lower, same, higher, not comparable (i.e., on a different branch of the tree). See Figure 3.

What specific connectionist network configuration should be inside the box is a research issue of its own. Further, what training algorithm should be used is also a topic for research. The larger the number of concepts in the knowledge base catalog, the more input nodes will be
required. The number of input nodes has direct impact on the amount of time it takes to train a network. For most current procedures, training time scales exponentially, or at best, polynomially, with number of network elements.

For the application here contemplated, once the catalog of concept types and the associated concept-type hierarchy for a knowledge base is developed, the connectionist network could be trained and used without subsequent modification. An unknown issue, however, is what happens when one more concepts is added to the catalog? Does the training algorithm allow for simple addition of one more piece of data with its relationships to the other data, or does the training process have to start over from scratch? Difficulties of this nature have been experienced [Rumelhart, McClelland & Williams, 1986]. The implications to training time are significant.

From the point of view of the present paper, the training problem, per se, is not the focus; rather, the focus is to argue the plausibility that a connectionist network can implement the desired operations on conceptual graphs. For this purpose, we assume that a connectionist network "black box" can be developed, and proceed to consider if and how it could be used to effect the PROPERTY 2 test for the PROJECTION operation in the data-base context.

We assume that information about which FactCGs passed the PROPERTY 1 test is available, and that only these FactCGs will be dealt with in this pass.

From the overall process point of view, it would be easiest if we were able to present the C-vector of the TemplCG (called Templ-C-vector) to the first group of input nodes (cf. Figure 3) and the C-vector of the current FactCG (Fact-C-vector) to the second group of input nodes, and then have the network (trained to) answer the following question: for each concept type with a 1 input in the first group, does there correspond at least one subtype of it with a 1 input in the

![Diagram](https://via.placeholder.com/150)

**Figure 3.** Neural net is trained to accept two RC-vectors, and to compare the two via the concept-type hierarchy.
second group? If the network could be trained to accomplish this procedure, then it would only need two outputs: one for yes, and one for no. This would solve the PROPERTY 2 test directly. But the likelihood of developing a connectionist network to solve as complex a problem as this is no doubt low at the present time. Even if the training-algorithm difficulties were to be solved for this kind of application, designing the set of inputs to be used for training the network so it is possible for the network to infer the task we want it to perform remains as an important problem. These are all subjects for research.

The more modest task we propose for the connectionist network is as follows (cf. Figure 3): present to the first group of input nodes one of the concept types which appears in the TemplCG (i.e., one of the active slots in the Templ-C-vector), and to the second group of input nodes, present one of the concept types which appears in the current FactCG (i.e., one of the active slots in the Fact-C-vector). Then, for this pair of activated terminals, have the connectionist network answer the (simpler) question regarding the location of the second concept type on the hierarchy relative to the first (e.g., lower, same, higher, not comparable). The results cited at the beginning of this section regarding family trees stand as evidence that the task suggested here should be doable via connectionist networks. There are a number of degrees of complexity in testing for PROPERTY 2 between the simple test suggested in this paragraph, and the much more complex one of testing for PROPERTY 2 directly. An example of a test between the two extremes so far suggested is to present to the first group of input nodes one of the concept types which appears in the TemplCG (as in the simplest test), and to the second group of input nodes, present the entire C-vector of the current FactCG (as in the most complex test), and then have the connectionist network answer the question whether there is at least one instance of the query concept type, or one of its subtypes, in the FactCG. Once the simplest test is successfully implemented, a progression of tests leading to the direct test could be researched.

Another aspect to consider is the case where a concept node has an individual marker assigned to it--i.e., the concept has been restricted (instantiated) to an individual. An example might be [person:Irene] or [man:Gregory]. When a concept node does have an individual marker assigned to it, a CONFORMITY test is required before performing certain RESTRICTIONs.

We assume a table is available which contains a list of all markers (names) used for individuals, and with each marker there is listed the concept subtype to which the marker normally applies.

Let us go through an example (cf. Figure 4). Suppose we are comparing [girl] with [person:Irene] and want to know if the RESTRICTION to [girl:Irene] is legitimate (i.e., if the concept-type referent of Irene CONFORMS to the concept type [girl]). If we check [girl] with [person] via the connectionist network of Figure 3, we find that [person] > [goal]. Now take into account that [person] is restricted to Irene; look up Irene in the marker list, and note that this marker has [female-person] as its referent. Check [female-person] with [girl] via the connectionist network to find that [female-person] > [girl]. Therefore, CONFORMITY holds, and [girl:Irene] is ok. On the other hand, suppose we are comparing [boy] with [person:Irene] and want to know if [boy:Irene] is legitimate. When we check [boy] with [person] via the connectionist network, we find that [person] > [boy]. As before, we now take into account that [person] is restricted to Irene; we look up Irene in the marker list, and again note that this marker has [female-person] as its referent. This time, when we check [female-person] with [boy] via the connectionist network, we find that [female-person] is not comparable to [boy]--i.e., [boy] is on a different branch of the tree. Therefore, CONFORMITY check fails, and we cannot make the RESTRICTION [boy:Irene].

This example demonstrates the potentially large number of accesses to the type/subtype checking operation that could be needed to perform the RESTRICTION & CONFORMITY check operations. The connectionist network makes the type/subtype operation possible in
constant time, no matter where in the hierarchy the pair being checked falls. Given such a potential time savings for each type/subtype check, the cumulative reduction of throughput time could be truly significant.

Figure 4. Names in the marker list include the concept-type referent for each marker. These are used to accomplish the CONFORMITY check. A marker can only be assigned to its referent concept type or to a concept type above or below it in the hierarchy. With the drawing as a visual aid, it is easy to see that the name Irene should not be assigned to the concept type boy (the referent concept type for Irene is said to not conform to the concept type boy). A suggested process for determining presence or absence of conformity via connectionist devices is given in text.

5. USING NEURAL NETS AS MEMORY FOR CONCEPTUAL GRAPHS

There is ample evidence in the connectionist literature [Hopfield, 1982] to indicate that vector patterns of the type defined above can be stored in connectionist networks. Further, there is evidence that associative and/or content addressing can be performed in such memories [Kohonen, 1984]. Conceptual graph applications would serve as a useful problem context for guiding research in using connectionist networks for memory. This is because the operations to be performed on conceptual graphs are becoming well defined for various types of applications [Sowa, et al, 1988], so researchers will have definite processing requirements to shoot for in developing connectionist memories. It could turn out, for example, that some of the PROPERTY-1 and PROPERTY-2 tests could be done implicitly in such memories. If so, then, for example, a TempICG could be specified for a query, represented appropriately, and presented to the input terminals of the connectionist memory. The output of this memory device could be, say, those FactCGs that have PROPERTY 1, and these are passed to the connectionist device that performs PROPERTY-2 tests. As the capabilities of connectionist devices are improved, perhaps it is not too far fetched to expect that one day, the connectionist memory can be trained to do the entire process.
6. CONCLUDING COMMENTS

There continue to be fundamental problems regarding training algorithms that need solving, so research along those lines is important to continue. At the same time, however, a way for one segment of researchers in this field to proceed would be to make an organized effort towards accumulating inventories of neural net architectures with knowledge of the kinds of problems each architecture is capable of solving—this requires studying constraints in the problem space from the point of view of comparing/matching them with the constraints in achievable mappings mentioned in the Introduction. We would then be in a position to study problem contexts, and conceptualize solution methodologies in terms of the problem-solving "skills" of the various neural network modules we have in our inventories. Networks could then be built up using the neural-net modules in a manner analogous to how program modules are used today in conventional programming—except that in the neural net case, activities will be going on in parallel. It seems to this author that this approach will hold a higher chance of success for implementing neural net computers to solve large, complex problems, than an approach which has a net attempt to learn an entire such problem directly.

It is with these ideas in mind, that the suggestion is made to mine the field of conceptual graph knowledge systems as a potentially rich source of (sub) problems that can be implemented with neural nets. Such an endeavor could benefit researchers in both fields.

REFERENCES


GGL
3/88