Path-Finding in Networks

Leixuan Yang and Stan Szpakowicz
Department of Computer Science, University of Ottawa
Ottawa, Ontario, Canada

Abstract

Algorithms that find connections between concepts in semantic networks are inherently inefficient due to their dependence on piecemeal representation of potential paths. A method of reorganizing a link-based network into a path-based representation is argued to allow more efficient search techniques. A path-based representation is presented in terms of the knowledge representation language Conceptual Network. Efficient path-finding algorithms based on this representation are described and analyzed. Search for a path connecting concepts in a network is turned into search among partial paths. The worst-case cost of searching for a simple path is quadratic; that for an extended path between families of closely related concepts is cubic.

1. Introduction

An essential reasoning task in network-like knowledge representation formalisms is finding paths between two or more concepts (Shapiro 1991, Selman & Levesque 1991). Quillian (1967) proposed manipulations on networks to make inference about pairs of concepts by finding connections between nodes that represent them. Finding a path can be reduced to the problem of link traversal (Shastri 1991). The existing path-finding methods take a start concept S and follow one of its alternative links to reach a goal concept G. The path is extended from every concept reached, until G has been found or the path cannot be further extended. If we cannot reach the second concept via an alternative, the search will backtrack to other alternatives. Various search strategies—versions of depth-first, breadth-first and heuristic search—could be applied. To limit the complexity of search, constraints on the path length can be imposed (Szpakowicz & Koperczak 1990), but then the solution

Proc. ICCI'94, 1042-1057

may not be found even if it exists. Parallel search methods have been proposed, such as the spreading activation method, first mentioned by Quillian (1967).

When people perform this kind of search, they do not find it difficult. For example, when one is shown a drawing of a network and asked to find a path between two given concepts, a solution will be found with little blind search or backtracking that the algorithmic methods require. This motivates us to emulate human performance to improve the efficiency of path-finding in networks.

We believe that the fundamental weakness of the existing methods is the representation of networks as collections of links between pairs of concepts. This divides the network into many small chunks, represented separately. A human, conversely, sees the network as a picture that can be regarded as a chunk of information much larger than a link between two concepts. This kind of *gestalt* may be what speeds up human search. Representing networks in this manner should improve the efficiency of reasoning.

We consider a path-based representation for networks, henceforth PB. A PB generator will transform the link-based representation of a network (knowledge base KB1) into the PB representation (knowledge base KB2). KB1 allows efficient reasoning in some circumstances, for example when seeking concepts directly related to a given concept. The original purpose of developing a path-based representation is to facilitate path-finding. As KB2 includes only the information necessary for constructing paths, it may have less information than KB1.

In section 2 of this paper we show how to represent hierarchical and non-hierarchical knowledge in PB. The discussion, couched in terms of our own network formalism, is kept general. We give two algorithms in section 3: finding a path of non-hierarchical links between concepts (links other than hierarchical relations *is_a* , *kind*) and finding a path that may consist of both hierarchical and non-hierarchical links between concepts. Conclusions are given in section 4.

2. Path-based representation of conceptual networks

In this section, we briefly introduce the PB representation of Conceptual Network, henceforth CN (Yang & Szpakowicz 1991a, 1991b). CN is the knowledge representation formalism in TANKA, a system for semi-automatic text analysis for knowledge acquisition (Szpakowicz 1990, Copeck *et al*. 1992, Delisle *et al*. 1993). TANKA is designed to process technical text and incrementally build a network which models the domain.

Four basic elements of CN are *concepts*, *relations*, *conditions* and *structures*. A *concept* represents a set of instances. I(*c*) denotes the set of instances of concept *c*. Concept *c1* is called a *subconcept* of concept *c2* (*c2* is called a *superconcept* of *c1*) iff I(*c1*) $\subseteq$ I(*c2*). There are three types of concepts: OBJECT, ACTIVITY and PROPERTY.

Relations capture relationships between concepts. CN only allows binary relations. Two hierarchical relations, specialization *is_a* and generalization *kind*, are used to build the conceptual hierarchies. For any two concepts *c1* and *c2*, is_a(*c1*, *c2*) iff *c1* is a subconcept of *c2*. For a binary relation *R* and two concepts *c1* and *c2*, *R(c1, c2*) means for any instance *i* of *c1*, there is an instance *j* of *c2* such that *i* relates to *j* by *R*.

Conditions represent contextual restrictions on a relationship. A condition attached to *R*(*c1*, *c2*) is regarded as the description of a subconcept *c1'* of *c1* and a subconcept *c2'* of *c2* such that only *c1'* and *c2'* can be in the relation *R*. A condition is built of simple conditions. A simple condition describes some instances of *c1* (or *c2*) by indicating their relations to other instances or to concepts which contain them. For example, the simple condition [location of block : table] describes a set of instances of *block*, {i | i $\in$ I(block) & j $\in$ I(table) & location(i, j) }, by relating them to instances of *table*. The simple condition [elephant : royal-elephant] describes a set of instances of *elephant*, {i | i $\in$ I(elephant) & i $\in$ I(royal-elephant)}, by naming the concept *royal-elephant* which contains them.

Structures are used to represent complex concepts, composed of other concepts. For example, in the "blocks world" the operation of *moving* a block can be decomposed into *picking-up* and *putting-down*. Moving a block can be represented as an ordered pair of operations:(tuple){picking-up, putting-down}.

In the following subsections we discuss the PB representation of hierarchical and non-hierarchical relations of CN, including those which are implied in conditions and structural relations. Other network formalisms can be analogously represented in PB.

2.1. Hierarchical relations

The conceptual hierarchies are the backbone of the network. There may be more than one hierarchy but every concept must be in a hierarchy. A *hierarchical graph* is associated with every concept *c*. It includes the superconcepts and subconcepts of *c*. The graph of *c* is written as:

$$\text{h\_graph}(c, [c_{i1}, c_{i2}, ..., c_{ik}], [c_{j1}, c_{j2}, ..., c_{jm}]).$$

where c_{ir}, r=1, 2, ..., k are the superconcepts of *c* and c_{jr}, r=1, 2, ..., m are the subconcepts of *c*.

Hierarchical graphs are used in extended search for paths, shown in detail in section 3.2.

2.2. Non-hierarchical relations

Besides hierarchical knowledge, networks contain knowledge about non-hierarchical relations between concepts. Such relations are represented either explicitly or implicitly in conditions attached to links in the network. The following kinds of relations between concepts are distinguished in CN:

1) Explicit relations without conditions. Such a relation is directly recorded in the network and no restriction is attached to it. The fact that concepts *c1* and *c2* are in such a relation *r* will be written in this paper as

link(*c1*, *c2*, *r*, expl).

2) Explicit relations with conditions. A relation of this kind is also directly recorded in the network, but a restriction is attached to it. The fact that concepts *c1* and *c2* are in such a relation *r* is written as

link(*c1*, *c2*, *r*, cond).

3) Structural relations. These are relations between complex concepts and their component concepts. Such relations are implicitly captured in the structural information of CN. The fact that concepts *c1* and *c2* are in such a relation *r* is written as

link(*c1*, *c2*, *r*, struct).

4) Implicit relations. Such relations between concepts are implied in the conditions of CN. A simple condition can be viewed as a connection between two concepts with more than one link. Concepts *c1* and *c2* related by a connection *t* may form such a link in a condition attached to a relation between concepts *c3* and *c4*. This is written as

$$\text{link}(c1,\ c2,\ t,\ \text{impl}(c3,\ c4)).$$

The fact that concept *c1* is in a relation with *c2* can be written as

$$\text{link}(c1,\ c2,\ r,\ \text{Relation\_type})$$

Non-hierarchical relations are represented in PB in a number of paths. Every path is a chunk of information that consists of a list of connected concepts, a list of relations between them and a list of labels which indicate whether the relations are explicit, implicit or structural. To facilitate access to individual concepts, relations or types of relations, the (1)format of a path in PB is:

$$\text{path}(\text{Path\_name},\ [c_1,\ c_2,\ ...,\ c_k],\ [r_1,\ r_2,\ ...,\ r_{k-1}],\ [t_1,\ t_2,\ ...,\ t_{k-1}]) \tag{1}$$

where c_i, i=1, 2, ..., k are concepts, r_i, i=1, 2, ..., k-1 are either relations or inverse relations, and t_i, i=1, 2, ..., k-1 are relation types (expl, cond, struct, or impl(_, _)).

For each pair c_i and c_{i+1}, i=1, 2, ..., k-1, the following link must be present in the network:

$$\text{link}(c_i,\ c_{i+1},\ r_i,\ t_i)$$

Concept c_1 in (1) is called the *start concept*, and c_k the *end concept* of the path. Both are called the *terminal concepts* of this path.

If the start and end concept of a path are identical, the path is called a *cycle.* If no concept occurs more than once in a path, it is called a *simple path*.

To avoid redundancy (by having every link in the network represented exactly once) and to simplify the solution of the path-finding problem, we impose two constraints on the path-based representation, which must be satisfied at the same time.

1) Every link in the network must be in one and only one path.
2) Every path must be either a cycle or a simple path.

To minimize the number of paths, we combine paths whenever possible. Let *p1* and *p2* be:

$$\text{path}(p1,\ [c_{i1},\ c_{i2},\ ...,\ c_{ik}],\ [r_{i1},\ r_{i2},\ ...,\ r_{ik-1}],\ [t_{i1},\ t_{i2},\ ...,\ t_{ik-1}])$$
$$\text{path}(p2,\ [c_{j1},\ c_{j2},\ ...,\ c_{jh}],\ [r_{j1},\ r_{j2},\ ...,\ r_{jh-1}],\ [t_{j1},\ t_{j2},\ ...,\ t_{jh-1}])$$

p1 and *p2* can be combined into one path *p*, which satisfies the second constraint, if:

1) c_{i1} is identical to c_{jh} and *p* is

$$\text{path}(p, [c_{j1}, \ldots, c_{jh}, c_{i2}, \ldots, c_{ik}],$$
$$[r_{j1}, \ldots, r_{jh\text{-}1}, r_{i1}, \ldots, r_{ik\text{-}1}], [t_{j1}, \ldots, t_{jh\text{-}1}, t_{i1}, \ldots, t_{ik\text{-}1}]),$$

2) c_{ik} is identical to c_{j1} and *p* is

$$\text{path}(p, [c_{i1}, \ldots, c_{ik}, c_{j2}, \ldots, c_{jh}],$$
$$[r_{i1}, \ldots, r_{ik\text{-}1}, r_{j1}, \ldots, r_{jh\text{-}1}], [t_{i1}, \ldots, t_{ik\text{-}1}, t_{j1}, \ldots, t_{jh\text{-}1}).$$

For example, suppose a network contains the following links:

link(*c1*, *c2*, *r1*, expl).
link(*c2*, *c3*, *r2*, impl(*cc1*, *cc2*)).
link(*c2*, *c5*, *r4*, cond).
link(*c3*, *c4*, *r3*, expl).
link(*c5*, *c6*, *r5*, struct).
link(*c5*, *c7*, *r6*, expl).
link(*c7*, *c8*, *r7*, impl(*cc4*, *cc8*)).

This network can be represented in PB by three paths in a way that satisfies constraints 1-2:

path(*p1*, [*c1*, *c2*, *c3*, *c4*], [*r1*, *r2*, *r3*], [expl, impl(*cc1*, *cc2*), expl]).
path(*p2*, [*c2*, *c5*, *c6*], [*r4*, *r5*], [cond, struct]). (2)
path(*p3*, [*c5*, *c7*, *c8*], [*r6*, *r7*], [expl, impl(*cc4*, *cc8*)]).

The PB representation for this kind of information is not unique: a different set of paths that satisfies the constraints may be generated to represent the same knowledge. For example, the above links can also be represented by these paths:

path(*pp1*, [*c1*, *c2*, *c5*, *c7*, *c8*], [*r1*, *r4*, *r6*, *r7*],
[expl, cond, expl, impl(*cc4*, *cc8*)]).
path(*pp2*, [*c2*, *c3*, *c4*], [*r2*, *r3*], [impl(*cc1*, *cc2*), expl]).
path(*pp3*, [*c5*, *c6*], [*r5*], [struct]).

Heuristic information based on types of concepts can help construct paths. Two activities can be connected by a common object—a role of both activities. Two objects can be connected by a common activity—two different roles in the same activity. Two properties can be connected by the same object—two different properties of the same object. Finally, an activity and a property can be related through a common object. The following kinds of path patterns will be chosen whenever possible:

1) ACTIVITY → OBJECT → ACTIVITY

2) OBJECT $\rightarrow$ ACTIVITY $\rightarrow$ OBJECT

3) PROPERTY $\rightarrow$ OBJECT $\rightarrow$ PROPERTY

4) PROPERTY $\rightarrow$ OBJECT $\rightarrow$ ACTIVITY

These could be used when extending a path: if there is more than one choice, consider the types of the last two concepts in this path, then apply the corresponding pattern to make a decision. For example, if these links are in the network:

link(*c1*, *c2*, *r1*, *t1*)
link(*c2*, *c3*, *r2*, *t2*)
link(*c2*, *c4*, *r3*, *t3*)

and the types of *c1*, *c2*, *c3*, *c4* are OBJECT, ACTIVITY, PROPERTY, OBJECT respectively, this network will be represented in PB by two paths:

path(*p1*, [*c1*, *c2*, *c4*], [*r1*, *r3*], [*t1*, *t3*])
path(*p2*, [*c2*, *c3*], [*r2*], [*t2*]).

Two extensions of *p1* are possible at *c2*: *c3* and *c4*. We choose *c4* because the types of *c1*, *c2*, *c4* match the pattern OBJECT $\rightarrow$ ACTIVITY $\rightarrow$ OBJECT.

3. Path-finding in the path-based representation

As we said in the introduction, finding connections between concepts is an essential reasoning task in networks, that can be turned into finding the paths connecting the given concepts in the network. PB was developed for this kind of reasoning task.

First, we define terms used in the presentation.

<u>Definition 1</u> A path *p* connects two concepts *c1* and *c2* iff *c1* and *c2* are the terminal concepts of *p*. This is written as connect-concept(*p*, *c1*, *c2*).

<u>Definition 2</u> Paths *p1* and *p2* intersect at concept *c* iff *c* occurs in both *p1* and *p2*. This is written as intersect(*p1*, *p2*, *c*).

<u>Definition 3</u> A path *p* connects two paths *p1* and *p2* iff there exist concept *c1* in *p1* and concept *c2* in *p2* such that connect-concept(*p*, *c1*, *c2*). This is written as connect-path(*p*, *p1*, *p2*).

It is easy to see that for any paths p, $p1$, $p2$, connect-path(p, $p1$, $p2$) iff there exist concepts $c1$ and $c2$ such that intersect(p, $p1$, $c1$) and intersect(p, $p2$, $c2$).

If concepts $c1$ and $c2$ occur in a path p, the subpath of p whose terminal concepts are $c1$ and $c2$ is denoted by subpath(p, $c1$, $c2$). Due to the second constraint, p is a simple path or a cycle. If p is a simple path, the subpath is unique. Otherwise, there must be two different subpaths between $c1$ and $c2$; the shorter subpath will be preferred. The cost of creating this subpath is O(M), where M is the number of concepts in the network. Because of the second constraint, the number of links in a path will not exceed M.

In the simplest case of finding the conceptual connections between two concepts there exists a path (consisting only of non-hierarchical relations) which connects these concepts. A more complicated case is when no such a path connects the given concepts, but they can be connected through some hierarchical relations; we must be able to perform extended search to find the paths which connect the hierarchical graphs of these two concepts.

3.1. Simple search

We first present a method of finding a path between two concepts in the network. Non-hierarchical knowledge in the network will be augmented with meta-knowledge of two kinds, stored in KB2 (see section 1). One is a list of paths indexed by concepts. The format of an index entry is

index(c, [p_1, p_2, ..., p_k]).

p_1, p_2, ..., p_k are the paths in which concept c occurs. For example, this is the index for the set of paths (2) in section 2.2:

index($c1$, [$p1$]).
index($c2$, [$p1$, $p2$]).
index($c3$, [$p1$]).
index($c4$, [$p1$]).
index($c5$, [$p2$, $p3$]).
index($c6$, [$p2$]).
index($c7$, [$p3$]).
index($c8$, [$p3$]).

The other kind of meta-knowledge in KB2 is represented in a matrix which we call the *intersection matrix* (denoted IM). Suppose there are N paths p_1, p_2, ..., p_N in the network.

IM is an $N \times N$ matrix. For every i, j such that $1 \le i, j \le N$, IM[i, j] = c_{ij} if intersect(p_i, p_j, c_{ij}), and IM[i, j] = **nil** otherwise.

We now present an algorithm for finding, for two given concepts *c1* and *c2*, a path *pp* such that connect-concept(*pp*, *c1*, *c2*). All the costs given for this algorithm are the worst-case costs.

<u>Algorithm 1</u> simple-path(*c1*, *c2*, *pp*)

Case 1.

If $\exists$ *p* such that there exists index(*c1*, *p11*) & index(*c2*, *p22*) &
$p \in p11 \cap p22$

then *pp* = subpath(*p*, *c1*, *c2*).

The cost of finding *p* is $O(N^2)$, and the cost of creating *pp* is O(M).

Case 2.

If $\exists$ *p1*, *p2* such that index(*c1*, p11) & index(*c2*, *p22*) &
$p1 \in p11$ & $p2 \in p22$ &
intersect(*p1*, *p2*, *c*)

then *pp* = subpath(*p1*, *c1*, *c*) $\oplus$ subpath(*p2*, *c*, *c2*).

($\oplus$ denotes concatenation)

The cost of finding *p1* and *p2* is $O(N^2)$ (checking in IM whether two paths intersect takes constant time), and the cost of creating *pp* is O(M), because the total number of links in *p1* and *p2* will not exceed 2M.

Case 3.

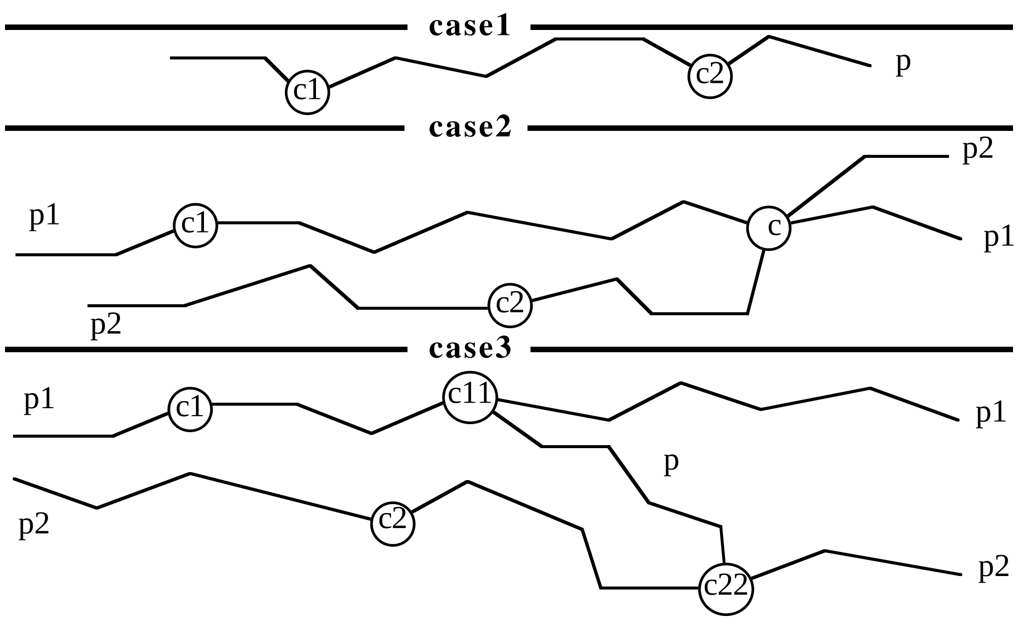
If $\exists$ *p1*, *p2*, *p* such that index(*c1*, *p11*) & index(*c2*, *p22*) &
$p1 \in p11$ & $p2 \in p22$ &
connect-path(*p*, *p1*, *p2*) &
intersect(*p1*, *p*, *c11*) & intersect(*p2*, *p*, *c22*)

then

pp = subpath(*p1*, *c1*, *c11*) $\oplus$ subpath(*p*, *c11*, *c22*) $\oplus$ subpath(*p2*, *c22*, *c2*).

The cost of finding a pair *p1* and *p2* is constant. Indeed, *p1* can be any path in *p11* and *p2* can be any path in *p22*: if there is a path between *c1* and *c2*, it can always be found in such a way with any *p1* and *p2*; that is to say, the cost does not depend on the choice of *p1* and *p2*. The cost of finding *p* is $O(N^2 + N*M)$, and the cost of creating *pp* is O(M). The total cost of the algorithm is $O(N^2 + N*M)$.

The three cases are illustrated as follows:



Finding a path *p* that connects paths *p1* and *p2* can be turned into finding a list of paths $pp_1, pp_2, ..., pp_k$ such that

intersect($p1, pp_1, c_{11}$) & intersect(pp_1, pp_2, c_2) &

……

intersect(pp_{k-1}, pp_k, c_k) & intersect($pp_k, p2, c_{22}$)

Then,

p = subpath(pp_1, c_{11}, c_2) $\oplus$ subpath(pp_2, c_2, c_3) $\oplus$ ... $\oplus$ subpath(pp_k, c_k, c_{22})

The cost of creating these subpaths from pp_1, ..., pp_k is O(N*M): the number of links in each path will not exceed M, the total number of links will never be greater than N*M. Finding the list pp_1, ..., pp_k is actually the same problem as finding a list of links that form a path between two nodes in a graph. There is no known algorithm for finding a path between two particular nodes that is more efficient in the worst case than Moore's Breadth First Search (BFS) technique or Dijkstra's algorithm. The complexity of both BFS and Dijkstra's algorithm is $O(K^2)$, where K is the number of the nodes in the graph—see the shortest path algorithms in Aho (1975), Sedgewick (1988) and Mehlhorn (1984). In our problem of finding a list of paths, intersections of paths are represented in the matrix IM, so that such a list can be found by the same method as finding a path in the graph. The cost is $O(N^2)$, where N is the number of paths or the size of IM.

3.2. Extended search

The simple search algorithm finds a path between two concepts, which contains only non-hierarchical relations. Such a path may not exist; if so, an *extended path* is sought as the solution.

We begin with definitions.

Definition 4 The *extended node* of concept *c*, written as ex-node(*c*), is the collection of concepts which are in the hierarchical graph of *c*.

Definition 5 An extended node ex-node(*c*) is in a path *p* iff there is a concept *c1* in ex-node(*c*) such that $c1 \in p$. This is written as exnode-in-path(*p*, *c*, *c1*).

Definition 6 A path *p* connects two extended nodes ex-node(*c1*) and ext-node(*c2*) iff there exists a concept c11 in ex-node(*c1*) and concept *c22* in ex-node(*c2*) such that connect-concept(*p*, *c11*, *c22*). This is written as connect-exnode(*p*, *c1*, *c2*),

Definition 7 An *extended path* is a list of paths $p_1, p_2, ..., p_k$, and a list of extended nodes ex-node(c_1), ex-node(c_2), ..., ext-node(c_{k+1}), such that
connect-exnode(p_1, c_1, c_2)
connect-exnode(p_2, c_2, c_3)
……

connect-exnode(p_k, c_k, c_{k+1})
ex-node(c_1) is called the *start node*, and ex-node(c_{k+1}) the *end node* of the extended path. Both nodes are called the *terminal nodes* of the extended path. An extended path is represented as ex-path([c_1, c_2, ..., c_{k+1}], [p_1, p_2, ..., p_k]).

Definition 8 An extended path with terminal nodes c_1, c_{k+1} connects two paths *pp1* and *pp2* iff there is a concept *cc1* in *pp1* and concept *cc2* in *pp2* such that $cc1 \in$ ex-node(c_1) & $cc2 \in$ ex-node(c_{k+1}). This is written as exconnect-path(p, $p1$, $p2$).

Definition 9 Two paths *p1* and *p2* intersect at an extended node ex-node(c) iff there exist concepts *c1* and *c2* in ex-node(c) such that *c1* is in *p1* and *c2* is in *p2*. This is written as ex-intersect($p1$, $p2$, c). We say that *p1* has an *extended intersection* with *p2*.

Suppose there is a path *p* such that

exnode-in-path(p, $c1$, $c11$) & exnode-in-path(p, $c2$, $c22$)

The subpath of *p*, subpath(p, $c11$, $c22$), is the path which connects ex-node($c1$) and ex-node($c2$). It is written as ex-subpath(p, $c1$, $c2$). Both subpath(p, $c11$, $c22$) and ex-subpath(p, $c1$, $c2$) are subpaths of *p*, but note that *c11* and *c22* are the terminal concepts of subpath(p, $c11$, $c22$), while *c1* and *c2* are not the terminal concepts of ex-subpath(p, $c1$, $c2$). What we know is that the terminal concepts of ex-subpath(p, $c1$, $c2$) are in ex-node($c1$) and ex-node($c2$).

The cost of checking whether a concept is in the extended node of another concept is O(M), so the cost of creating ex-subpath(p, $c1$, $c2$) is $O(M^2)$.

When the simple search cannot find a path connecting the given concepts *c1* and *c2*, an extended path with ex-node($c1$) and ex-node($c2$) as its terminal nodes is sought. From this extended path, the conceptual connection between *c1* and *c2* can be easily obtained. Just as we use meta-knowledge (the index and the intersection matrix) in knowledge base KB2 for the simple search, we have analogous meta-knowledge (an index for the extended nodes, and a matrix for the ex-intersect relations) for the extended search.

There is an index entry for every extended node:

ex-index(c, [p_1, p_2, ..., p_k])

where p_1, p_2, ..., p_k are the paths in which ex-node(c) occurs. That is to say, for every p_i, i=1, 2, ..., k, there exists c_i such that exnode-in-path(p_i, c, c_i).

An $N \times N$ matrix called the *extended intersection matrix* (denoted EIM) describes the ex-intersect relations between paths. For every $1 \leq i, j \leq N$, EIM[i, j] = c_{ij} if ex-intersect(p_i, p_j, c_{ij}), and EIM[i, j] = **nil** otherwise.

The following algorithm finds an extended path with ex-node(*c1*) and ex-node(*c2*) as its terminal nodes. Actually, the methods for both simple and extended search problems are the same. Their analogy can be observed from the similarity of algorithm 1 and algorithm 2. All the costs stated for algorithm 2 are the worst-case costs.

<u>Algorithm 2</u> extended-path(*c1*, *c2*, *extp*)

Case 1.

If ∃ *p* such that	ex-index(*c1*, *p11*) & ex-index(*c2*, *p22*) &
	p ∈ *p11* ∩ *p22*
then	*extp* = ex-path([*c1*, *c2*], [*p33*])
where	*p33* = ex-subpath(*p*, *c1*, *c2*).

The cost of finding *p* is $O(N^2)$. The cost of creating extp is $O(M^2)$.

Case 2.

If ∃ *p1*, *p2* such that	ex-index(*c1*, p11) & ex-index(*c2*, *p22*) &
	p1 ∈ *p11* & *p2* ∈ *p22* &
	ex-intersect(*p1*, *p2*, *c*)
then	*extp* = ex-path([*c1*, *c*, *c2*], [*p33*, *p44*])
where	*p33* = ex-subpath(*p1*, *c1*, *c*),
	p44 = ex-subpath(*p2*, *c*, *c2*).

The cost of finding *p1* and *p2* is $O(N^2)$ (checking in EIM if one path has an extended intersection with another path takes constant time) . The cost of creating extp is $O(M^2)$, because the total number of links in *p1* and *p2* will not exceed 2M.

Case 3.

If ∃ *p1*, *p2*, *p* such that	ex-index(*c1*, p11) & ex-index(*c2*, *p22*) &
	p1 ∈ *p11* & *p2* ∈ *p22* &

p = ex-path([cp_1, cp_2, ..., cp_k], [pi_1, pi_2, ..., pi_{k-1}]) &
exconnect-path(p, $p1$, $p2$) &
ex-intersect($p1$, pi_1, $c11$) &
ex-intersect($p2$, pi_{k-1}, $c22$)

then *extp* = ex-path([$c1$, $c11$, cp_2, ..., cp_{k-1}, $c22$, $c2$],
[$p33$, $p44$, pi_2, ..., pi_{k-2}, $p55$, $p66$])

where $p33$ = ex-subpath($p1$, $c1$, $c11$),
$p44$ = ex-subpath(pi_1, $c11$, $cp2$),
$p55$ = ex-subpath(pi_{k-1}, cp_{k-1}, $c22$),
$p66$ = ex-subpath($p2$, $c22$, $c2$).

The cost of finding a pair $p1$, $p2$ is constant: $p1$ can be any path in $p11$ and $p2$ can be any path in $p22$. Indeed, let there be an extended path between $c1$ and $c2$; it can always be found in the way described in algorithm 2, that is, it does not depend on the selection of $p1$ and $p2$. The cost of finding p will be shown below to be $O(N^2 + N*M^2)$, and the cost of creating extp is $O(M^2)$. The total cost of the algorithm is $O(N^2 + N*M^2)$.

The three cases described above could be illustrated as previously, except that the symbols in circles would now correspond to extended rather than simple nodes, and path p would be an extended path.

Finding an extended path p which connects paths $p1$ and $p2$ (which is similar to finding a path which connects two paths), can be turned into finding a list of paths pp_1, pp_2, ..., pp_k such that

ex-intersect($p1$, pp_1, c_1) &
ex-intersect(pp_1, pp_2, c_2) &
……
ex-intersect(pp_{k-1}, pp_k, c_k) &
ex-intersect(pp_k, $p2$, c_{k+1})

Let

pi_1 = ex-subpath(pp_1, c_1, c_2),
pi_2 = ex-subpath(pp_2, c_2, c_3),
……
pi_k = ex-subpath(pp_k, c_k, c_{k+1}).

Then p = ex-path($[c_1, c_2, ..., c_{k+1}], [pi_1, pi_2, ..., pi_k]$).

The cost of creating the subpaths $pi_1, pi_2, ..., pi_k$ from $pp_1, pp_2, ..., pp_k$ is O(N*M^2): the cost of checking if a concept is in the extended node of another concept is O(M), and the total number of links in these path $pp_1, pp_2, .., pp_k$ will never be greater than N*M. Actually, finding a list of paths $pp_1, pp_2, ..., pp_k$ is, once more, the same problem as finding a list of links to form a path between two nodes in a graph. Extended intersection of paths is represented in the matrix EIM, so such a list of paths can also be found by the same method as finding a path in a graph. The cost is O(N^2), where N is the number of paths or the size of EIM.

4. Conclusion

We have presented a path-based (PB) representation for finding connections between concepts in a network. PB is a possible way of improving the complexity of the path finding problem. In PB, the complexity of the simple search problem (finding a connection between concepts which consists of only non-hierarchical links) is O(N^2+N*M), where N is the number of paths in the network and M is the number of concepts in the network. Because N is never greater than M, this complexity will never exceed O(M^2). When N compared to M is very small, the complexity of our simple search will be close to linear. The complexity of extended search to find a connection between concepts which may consist of hierarchical links is O(N^2 + N*M^2), which will never exceed O(M^3).

The path which is found to connect two concepts may be meaningless from the semantic point of view, because few semantic relations between close concepts are considered here, and the creation of those paths is almost mechanistic. To create more meaningful paths, additional heuristic information of the kind mentioned in section 2.2 will be required.

References

Aho, A. V., J. E. Hopcroft & J. D. Ullman (1975) *The Design and Analysis of Algorithms*, Addison-Wesley.

Copeck, T., S. Delisle & S. Szpakowicz (1992) “Parsing and Case Analysis in TANKA”. *Proc 15th Intl Conf on Computational Linguistics COLING-92*, Nantes, 1008-1012.

Delisle, S., T. Copeck, S. Szpakowicz & K. Barker (1993) "Pattern Matching for Case Analysis: A Computational Definition of Closeness". O. Abou-Rabia, C. K. Chang, W. W. Koczkodaj (eds.) *Proc ICCI-93*, 310-315.

Mehlhorn, K. (1984) *Data Structures and Algorithms 2: NP-completeness and Graph Algorithms*, Springer-Verlag, Berlin.

Quillian, M. R. (1967) “Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities”. *Behavioral Science* **12**, 410-430.

Sedgewick, R. (1988) *Algorithms*, Addison-Wesley.

Selman, B. & H. J. Levesque (1991) “The Tractability of Path-based inheritance”. In (Sowa 1991), 283-297.

Shapiro, S. C. (1991) “Cables, Paths, and Subconscious Reasoning in Propositional Semantic Networks”. In (Sowa 1991), 137-156.

Shastri, L. (1991) “Why Semantic Networks”. In (Sowa 1991), 109-136.

Sowa, J. F. (ed.) (1991)*Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Morgan Kaufmann.

Szpakowicz, S. (1990) “Semi-Automatic Acquisition of Conceptual Structure from Technical Texts”. *Int J of Man-Machine Studies*, **33**, 385-397.

Szpakowicz, S. & Z. Koperczak (1990) “Mixed-Strategy Matching in Conceptual Networks”. Z. W. Ras, M. Zemankowa & M. L. Emrich (eds.) *Methodologies for Intelligent Systems 5.* North-Holland, 321-328.

Yang, L. & S. Szpakowicz (1991a) “Planning in Conceptual Networks”. F. Dehne, F. Fiala & W. W. Koczkodaj (eds.) *Advances in Computing and Information - ICCI'91. Lecture Notes in Computer Science*, **497**, Springer-Verlag, 669-671.

Yang, L. & S. Szpakowicz (1991b) “Inheritance in Conceptual Networks”. Karen S. Harber (ed.) *Proc Sixth Int Symposium on Methodologies for Intelligent Systems (Poster Session)*. Charlotte, NC, Oct. 1991, 191-202.