Recovering Problem-Solving Activities from Query Messages

Yoshihiko HAYASHI
NTT Information and Communication Systems Laboratories
1-2356 Take, Yokosuka, 238-03 JAPAN
E-mail: ha.yashi@nttnly.isl.ntt.jp

Abstract

We investigated a set of query messages taken from an Usenet newsgroup, and analyzed relations between the nature of problem-solving activities and their natural language descriptions. Based on the corpus investigation, this paper proposes an efficient computational mechanism for recovering problem-solving activities from query messages written in Japanese. The main claim of the paper is that the underlying problem-solving activity described in a natural language message can be efficiently recovered if provided with general knowledge on human problem-solving and the associated linguistic patterns in the descriptions.

1 Introduction

In this paper, we focus on *trouble-shooting* type problem-solving activities. One general way to solve such a complex problem is to know a solution which might solve the problem and to apply it. These days some people utilize computer networks as a forum useful for problcm-solving[llammond, 1995]. Thus understanding how people ask for information with electronic messages will be inevitable to realize an intelligent information access function on information super highways.

With this motivation, we started this work with an investigation of a set. of actual query messages, which had been taken from an Usenet newsgroup[Krol, 1993]. The newsgroup chosen as our corpus is "fj.sys.mac"[1], which provides a forum of discussions about Apple's Macintoshes, mainly for Japanese-speaking people. We selected forty-two messages, aimed at trouble-shooting, out of 119 query messages which appeared in a three-months' term in 1993. Each of these selected messages contains a description of the writer's trouble-shooting type problem-solving activity.

2 A Motivational Example

Let us start with a translated and slightly simplified example of an actual query message taken from the corpus.

[1] Needless to say, there are no particular intentions for the choice.

(Example)
S1: I found my MacII had been infected with some kind of virus.
s2: So, I wanted to run a vaccine program and
s3: tried to make my Mac read an MS-DOS formatted 2DD floppy disk.
s4: However, Apple File Exchanger could not read it.
s5: Can't MacIIs read MS-DOS formatted 2DDs?

The story begins with his/her discovery of a trouble (si) and ends with a query (s5). Sentences s2 through s4 describe his/her trouble-shooting type problem-solving activity. How can people read this message? From s2, we will be able to guess that, at least for him/her, running a vaccine program is a way to solve the trouble stated in si. Thanks to the marker "so" in s2, we will be able to guess this, even if we do not have good knowledge about disinfecting computer viruses.

On the other hand, the relation between s2 and s3 seems to be unclear on two points. First, because the two sentences are connected by the conjunction "and", there are two possibilities for interpreting the role of the action described in s3; that is, either *running a. vaccine program followed by making Mac read a 2DD disk will shoot the trouble,* or to *run* a vaccine *program, making Mac read* a *2DD disk is necessary.* Main verbs in both sentences may provide a slight clue. However, it does not seem to be a strong one. The domain knowledge, in general, might help the disambiguation process. However, if his/her actions were based on his/her wrong belief or knowledge, even correct domain knowledge would not be of help[2]. In either case, it is clearly shown by S4 that his/her trial to make Mac read a 2DD disk failed and this suggests that he/she still cannot disinfect the virus from his/her MacII.

Note that the entire problem-solving process described in the message can be seen as a goal-directed activity. The trouble stated in si may prevent him/her from achieving his/her primary goal, which is not explicit in the message. Therefore he/she introduces a new goal, disinfecting the virus. The activities described in s2 through s4 are associated with the new goal. In addi-

[2] In this example, his/her doing s3 is actually underivable from correct domain knowledge. Most of the response messages to this query have pointed it out.

tion, the query stated by s5 can be seen as a step of the problem-solving, although the level differs from the actual activities.

3 A Recovery Mechanism

3.1 Overview

Our recovery mechanism consists of three cascaded processes. The first process identifies message fragments, each of which describes a single problem-solving step of the entire activity, by recognizing some distinctive surface expression patterns, as in many information extraction systems[Hobbs, 1992],[Kitani, 1994], Each fragment is then assigned a sub-graph which represents the structure of the step by the second process. The third process finally integrates these sub-graphs and represents the entire problem-solving activity as a set of graph structures with help from general knowledge on problem-solving and the associated linguistic heuristics in the discourse structures. Each of the graphs represents a possible underlying problem-solving activity. We call such a graph problem-solving graph, or PS-graph in short.

3.2 Ontology for the PS graphs

In this paper, we assume that a goal state is enabled by one of the alternative goal-achieving actions, and a goal-achieving action is generated by a sequence of decomposed actions.

Table 1: Links in the PS-graphs.

Type	Link	Source Node	Destination Node
Action	alternative	X	G
	enable	X	G
	a-step	X	X
	generate	X	X
	non-executable	P	X
Causality	caused	X	P
	cause-always	X	P
	prevent	P	X
Problem -Solving	to-solve	G	P
Auxiliary	similar	P	P

There are four node types for the PS-graphs;

- G node: It encodes a goal state of the agent (writer), and may have its value, achieved/unachieved.
- P node: It encodes that a problematic situation (a trouble) came about, and may have its value, fixed/unfixed.
- X node: It encodes an action by the agent, and may have its value, succeeded/failed.
- XP node: It is a composite node by an X node and a P node connected by a causality type link. It encodes a causal relation, an action caused a trouble.

We have ten direct link types as shown in Table 1.

- Action type: *alternative* link encodes the action X is one of the alternative actions which enables the goal G. *Enable* link is used in cases where X is the only action to enable the goal. *A-step* link encodes the action X is one of the decomposed action to achieve the goal G. Generate link is used in cases where X is the only decomposed action. *Not-executable* encodes the problem P prevents execution of the action X.
- Causality type: Caused link encodes the action X caused the problem P. *Cause-always* link is used when the agent knows a fact that X always causes P which may be an undesirable side-effect.
- Problem-solving type: *To-solve* link encodes the agent has a new goal state G in which the problem P does not come about. It is automatically introduced whenever the agent knows a problem came about.
- Auxiliary type: *Similar* link connects two same/similar problematic states. It is typically used in cases where the same problem comes about again, in spite of the fact that a problem-solving activity has been conducted.

4 Recovery of Single PS Step

4.1 Types of problems

We looked through the forty-two objective messages and extracted sixty-three problem description fragments. Table 2 classifies types of the problems. The classification was conducted by focusing on following three points:

1. Did he intend to do some action?
2. Did he actually do the action?
3. Is there a problematic situation /si now?

In the table, example expressions in English and the number of occurrences for each type are also shown. Here, we distinguished two situations:

- FI (primary): The problem is introduced as a primary problem for the writer. Because most of the problem-solving activities are described in chronological order, it appears prior to the other levels of problems in the descriptions.
- F2 (lower-level): The problem came about during the problem-solving activities. It mainly describes a situation where the writer's goal-achieving or problem-solving activity caused the same problem or introduced a new problem.

Note that we included the problem type (f) in our variety of problem types. In this type, actually, there is no trouble anymore. For the example in the table, the writer fixed the problem *(the print process doesn't terminate)* by an action *(send Ctrl-d to the printer).* However, we can imagine that he/she is not convinced by the solution, and wants to know a better solution. We see human problem-solving in a broader sense than usual.

4.2 Identifying problem descriptions

Table 3 lists expression patterns which can be used for the identification of problem descriptions. By these patterns, 92.1% (58/63) of the problem descriptions in the

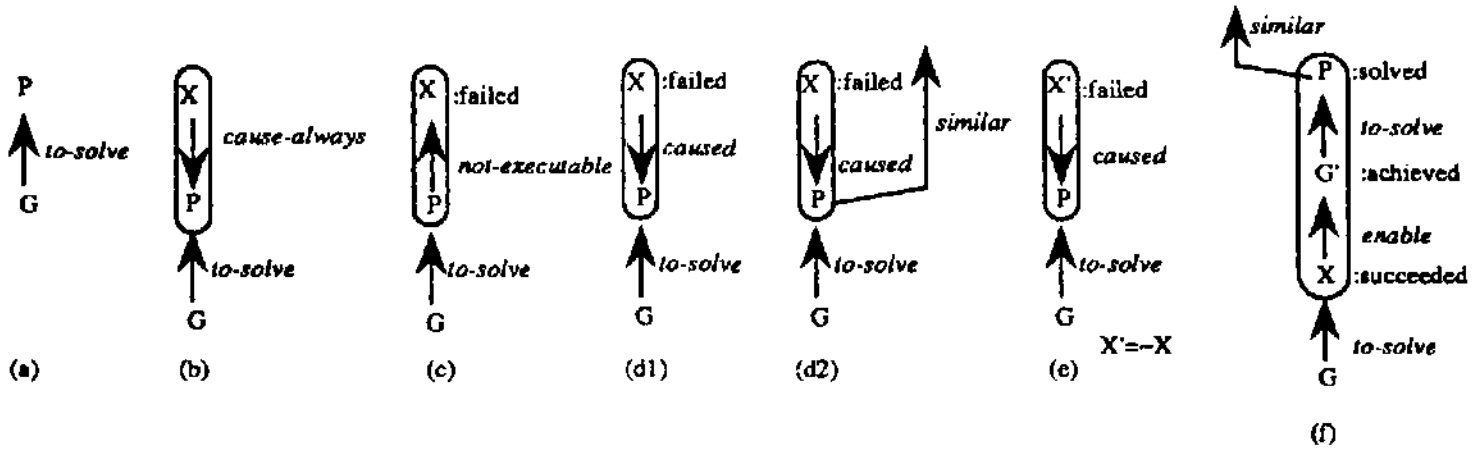

Figure 1: Sub-graphs for the problem types.

Table 2: Types of problems.

Type	1	2	3	F1	F2	Example
a: Unexpected trouble	-	-	-	11	2	I found my system had been infected by a virus.
b: Undesirable side-effect	+	-	-	3	0	If we access a file on the server, a warning appears.
c: Impossible action	+	-	-	6	2	I tried to connect it, but I don't have the connection kit.
d1: Action failure	+	+	+	19	11	When I sent a mail by it, the file code was changed.
d2: PS action failure	+	+	+	0	5	Even if I reinstalled the system, same error occurred.
e: Trouble avoidance failure	+	-	+	2	0	I didn't select the File-Share, but it tried to connect.
f: Undesirable trouble fix	+	+	-	1	1	Unless we send Ctrl-d, the print process doesn't terminate.

corpus are covered. Particular modal expressions in Type-A provide the strongest clues; and these are *closed* expressions.

Here we would like to remark on two major modal expressions which strongly suggest that the associated propositional part of a description describes a problematic situation.

- verb + "te-shimau": This expression's original function is to represent the perfect tense. However in many cases, it represents a nuance that the writer thinks/feels that the situation described by the main-clause is undesirable for him/her[Masuoka, 1989]. In this domain, it turns out to be describing a problematic situation.
- verb + "te-kure-nai": This is a negation form of "te-kureru", which represents a situation where the agent of the verb is not the writer and the associated situation is desirable *(empathy* in [Kuno, 1987]) for him/her. Therefore, in this domain, its negation form, "te-kure-nai", represents that the writer thinks the situation caused by a systems/program's behavior is problematic.

In contrast to these expressions, other Types are *open expressions.* Thus, we must prepare a set of explicit patterns which covers possible expressions in advance.

4.3 Recognizing problem type

The problem types (b) through (f), shown in Table 2, are expressed mainly by conditional clauses/sentences. Table 4 lists combinations of expression patterns by which we can map an identified problem description into one of the problem types. In the table, we indicate the patterns as a triple, x-part, connective, and y-part; here, x-part is the antecedent and y-part is the consequent of a conditional expression. The following are brief explanations about the distinctive Japanese connectives which appear in the table with respect to the problem types.

- "-baai" (= in case of -ing): Use of this connective suggests that the conditional sentence describes a kind of fact. Thus, it insists that the problem type may be (b), in which an undesirable side-effect caused by an action is described.
- "-nodesuga" : This connective establishes the antecedent (x-part) as a background description. Use of this connective, together with an intentional expression in the x-part, suggests to us that one of the pre-conditions for the intended action did not hold at the time.
- "-to" (= if/when): This connective is the most common and neutral one. It corresponds to "if* or "when" in English.
- "-noni/nimo kakawarazu" (= even if): These connectives might be similar to "even if'' or "although" in English. As [Masuoka, 1989] discusses, these connectives emphasize that there is an expectation violation for the speaker. Therefore, this type is strongly connected with the problem type (d2) and (e).

We confirmed that the mapping rule gave us a fairly accurate result (average-recall: 73.0%, average-precision: 90.2%) toward the corpus data by a hand simulation. As the result shows, in the current mapping rule, stress is placed on the precision rate.

Table 3: Surface expression patterns in problem descriptions.

Type	Example	Freq
A: Modal expression	zokusei ga kawat te-shimau attribute SUBJ change MODAL The attribute has changed.	29
B: It does not work	sono sousa ha umakuika nai the operation TOPIC work NEG The operation doesn't work.	8
C: Error occurs	eraa-hyouji ga araware ta error-message SUBJ appear PAST An error message was displayed.	11
D: Don't know	houhou ga wakara nai method SUBJ know NEG I have no idea about the method.	5
E: System crashes	shisutemu wo buuto deki nai system OBJ boot can NEG I cannot boot the system.	3
F: Unexpected system behavior	yosou-ni-hanshite tadashiku hyouji sare nai unexpectedly properly display PASSIVE NEG Unfortunately, it is not displayed properly.	2

Table 4: Mapping to problem types from surface patterns.

x-part	Connective	y-part	Type
-	-	A,C,D,E,F	(a)
present + positive	-baai	(general)	(b)
intentional verb	-nodesuga	B,D	(c)
present + positive	-to	A,B,C,E,F	(d1)
past + positive	-nomi/nimo+kakawarazu	A,B,C,E,F	(d2)
negative	-nomi/nimo+kakawarazu	A,B,C,E,F	(e)
negative	-to	A,B,C,E,F	(f)

4.4 Assigning a sub-graph

By the second process of the mechanism, each of the extracted fragments is then assigned one of the ready-made sub-graphs shown in Figure 1 according to its problem type. Each situation and the associated newly introduced goal G can be described as follows:

- type (a): He/she has a new goal state G in which the problem encoded in P is fixed.
- type (b): He/she knows that there is an undesirable side-effect represented by P. The new goal G is either, execute X without P, or simply solve P.
- type (c): He/she knows that one of the preconditions for executing X did not hold. Hence, he/she believes that X is not executable. To make hold the condition and make X be executable is the new goal.
- type (dl): He/she knows that the action X caused a problematic situation P, and has a goal to solve it. The problem P would prevent achievement of upper-level goals.
- type (d2): He/she knows that the action X, intended to solve an upper-level problem, caused same problem P again; and he/she has a new goal to solve it.
- type (e): [Allen, 1984] introduced a predicate *NOT-T0-D0(a,t,p)* it would be true if the plan p includes not performing the action a at the time t. Like this treatment, we regard not performing the action X intentionally as a kind of action. Therefore, we link the X' and the P node by the *caused* link. We can imagine, in this case, that the writer is wondering why the solution didn't work. The new goal G, thus, would be to know the reasons, and the associated information request may be described in the query message.
- type (f): As mentioned in 4.1, the sub-graph encodes an extended problem-solving activity. The newly introduced goal G would be to know a better solution to solve the once solved problem P. That is, the G would be strongly associated with an information request in the query message.

5 Recovery of an Entire PS Activity

5.1 Overall algorithm

Figure 2 outlines the overall algorithm. The input to the algorithm is a set of sub-graphs provided by the second process. As far as we looked through the set of messages, most of the problem-solving activities are described in chronological order. Thus, the algorithm basically iter-

ates through the each sub-graph in the input set, and constructs a set of possible PS-graphs incrementally [3].

5.2 Linking sub-graphs

The Base-line

The central part of the algorithm links a current sub-graph (c-graph in the algorithm) into a PS-graph being constructed. The procedure, *make-graph-1,* does the job. The crucial point here is which node in the c-graph should be connected to which node in the being constructed PS-graph by which type of link.

The answer to the first point is provided by the procedure, *get-top-node.* It simply returns the top node in the c-graph as one part of the connection point. Note that we can naturally expect that the returned node is limited to the action type (X node). As indicated in Figure 1, all the top nodes are X nodes, except for problem type (a) and (f). Because the problem type (a) only appears in the beginning of descriptions, we don't have to worry about this. In addition, the problem type (f) can be seen as an extension to the main problem-solving activity; the top P node would be simply linked to another P node by a similar link [4].

The answer to the second point is given by the procedure, *get-link-points.* Given a PS-graph being constructed, it returns possible nodes in the PS-graph which can be linked to. According to the graph ontology introduced by Table 1, G nodes, X nodes and P nodes would be the candidates. However, we can exclude P nodes, as these are provided only by the input to the algorithm. They are already pointed by X nodes, or they are top-level nodes.

The answer to the third point is provided by the procedure, *check-linkable.* Given a pair of nodes which consists a connection point, it returns a type of link to be used. Based on the discussion so far, a possible pair {PS-graph-node, c-graph-nodo} is either {G node, X node} or {X node, X node}. As enable/generate links are special cases of alternative/a-stcp links respectively, the procedure returns *alternative* for the first pair, and *a-step* for the second pair.

Constraints and heuristics for reducing the candidates

The *get-link-points* may return more than one node in a PS-graph being constructed as a candidate of the linking points. As this nature is a source of the ambiguities in PS-graphs, it is important to reduce the number of the candidates to avoid the combinatorial explosion.

We can utilize some constraints to exclude inappropriate candidates. First, we cannot select the X nodes which are already marked *failed* as a candidate: the agent cannot perform any action in order to perform an already failed action. Second, we cannot choose the G nodes which are already marked *achieved* as a candidate. Such G nodes are possible, if the (f) type sub-graph has already been introduced to the PS-graph. These constraints seem to naturally come from general schema of

[5]For the explanation, the algorithm is slightly simplified. There would be several ways to optimize it.

We simply ignore this case in the rest of the paper.

```
1. procedure recover-entire-PS-activity(sub-graphs)
2. make-graphs({car(sub-graphs)},cdr(sub-graphs));
3. end procedure;

1. procedure make-graphs(PS-graphs,rest-graphs)
2. local-variables begin rslt-graphs:={}; end;
3. if rest-graphs=={} then
     return(assign-values(reduce-graphs(PS-graphs)));
4. for each PS-graph in PS-graphs do
5.   rslt-graphs:=rslt-graphs ∪
       make-graphs-1(PS-graph,car(rest-graphs));
6. make-graphs(rslt-graphs,cdr(rest-graphs));
7. end procedure;

1. procedure make-graphs-1(PS-graph,c-graph)
2. local-variables begin
3.   rslt-graphs:={};
     link-points:=get-link-points(PS-graph);
     link-relations:={};
4. end;
5. if link-points=={} then signal('link-error-1');
6. for link-point in link-points do begin
7.   link-relation:=check-linkable(link-point,
                          get-top-node(c-graph));
8.  if link-relation==Φ then signal('link-error-2');
9.   rslt-graphs:= rslt-graphs ∪
              link-graphs(PS-graph,link-point,
                          c-graph,link-relation);
10. end;
11. return(rslt-graphs);
12. end procedure;
```

Figure 2: PS-graph construction algorithm.

human problem-solving activities. Third, as a PS-graph is constructed as top-down and left-to-right with time, we can use this constraint to reduce the candidates.

Furthermore, we can introduce some heuristics to prune unlikely candidates. The following are principles [5] which also come from general schema of human goal-achieving activities:

- *Principle of alternative* actions: Suppose X\ through X_n are alternative actions to enable a goal G, and are sorted by goodness of the actions. It is natural to assume that the agent will try to perform A'_{I+1} only when he/she failed to perform A",-.
- *Principle of decomposed actions:* Suppose A'i through X_n form a sequence of the decomposed actions to generate an action X. It is natural to assume that the agent won't try to perform X_{i+1} if he/she failed to perform X_t.

With these principles, the associated heuristics can be summarized as follows: In a situation like Figure 3(a), linking the c-graph to G is allowed only when the *Xi* is already marked *failed.* That is, failure of *X{* must be represented in the description. The G nodes which violate this cannot be the candidates. On the other hand, if there is a parallel relation in the associated discourse structure, the Unking is strongly preferred.

In a situation like Figure 3(b), linking the c-graph to X'o is allowed only when the X_i is not already marked *failed.* The X nodes which violate this cannot be the

[6]Of course, these principles are too strong. However, we confirmed that most of the messages follow these principles.

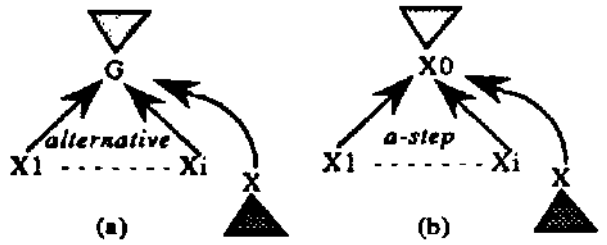

Figure 3: Possible linking situations.

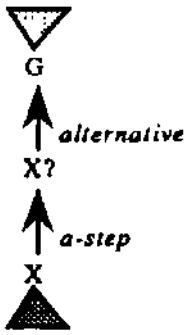

Figure 4: Graph with a virtual node.

candidates. On the other hand, if there is a temporal/sequential relation in the associated discourse structure, the linking is strongly preferred.

Linking sub-graphs actually

Given a PS-graph being constructed, a linkable node in the PS-graph, a c-graph and a possible link type, the procedure *link-graphs* actually links the c-graph into the PS-graph. Figure 3 has already shown how the linking would be done.

However, note that when a c-graph is linked to a G node in a PS-graph, another graph, as shown in Figure 4, must be generated. As mentioned in 3.2, we assume that a goal state is enabled by one of the alternative goal-enabling actions, not directly by a sequence of decomposed actions. But, these goal-enabling actions are often missed in the natural language descriptions. Therefore, to handle such a case, we must assume the existence of a virtual goal-enabling action and incorporate it in the PS-graph. We call such a virtual goal-enabling action virtual action, and denote it as X? in the PS-graphs.

5.3 Post-process

The following are conducted as post-process.

Reducing redundant PS-graphs (reduce-graphs): As enable/generate links are special cases of alternative/a-step links, this process first checks the alternative/a-step links that do not have sister links and changes them to enable/generate links. As often redundant PS-graphs are constructed due to the virtual action nodes, the procedure next checks redundant graphs, and preserves the most simple ones only.

Assigning success/failure values to nodes (assign-values): Based on the principles of alternative actions and decomposed actions, we can assign *succeeded/failed* to action nodes, *achieved/unachieved* to G nodes and *fixed/unfixed* to P nodes in a PS-graph, by propagating explicitly stated success/failure values, which come from the sub-graphs in the input set, along the directed links.

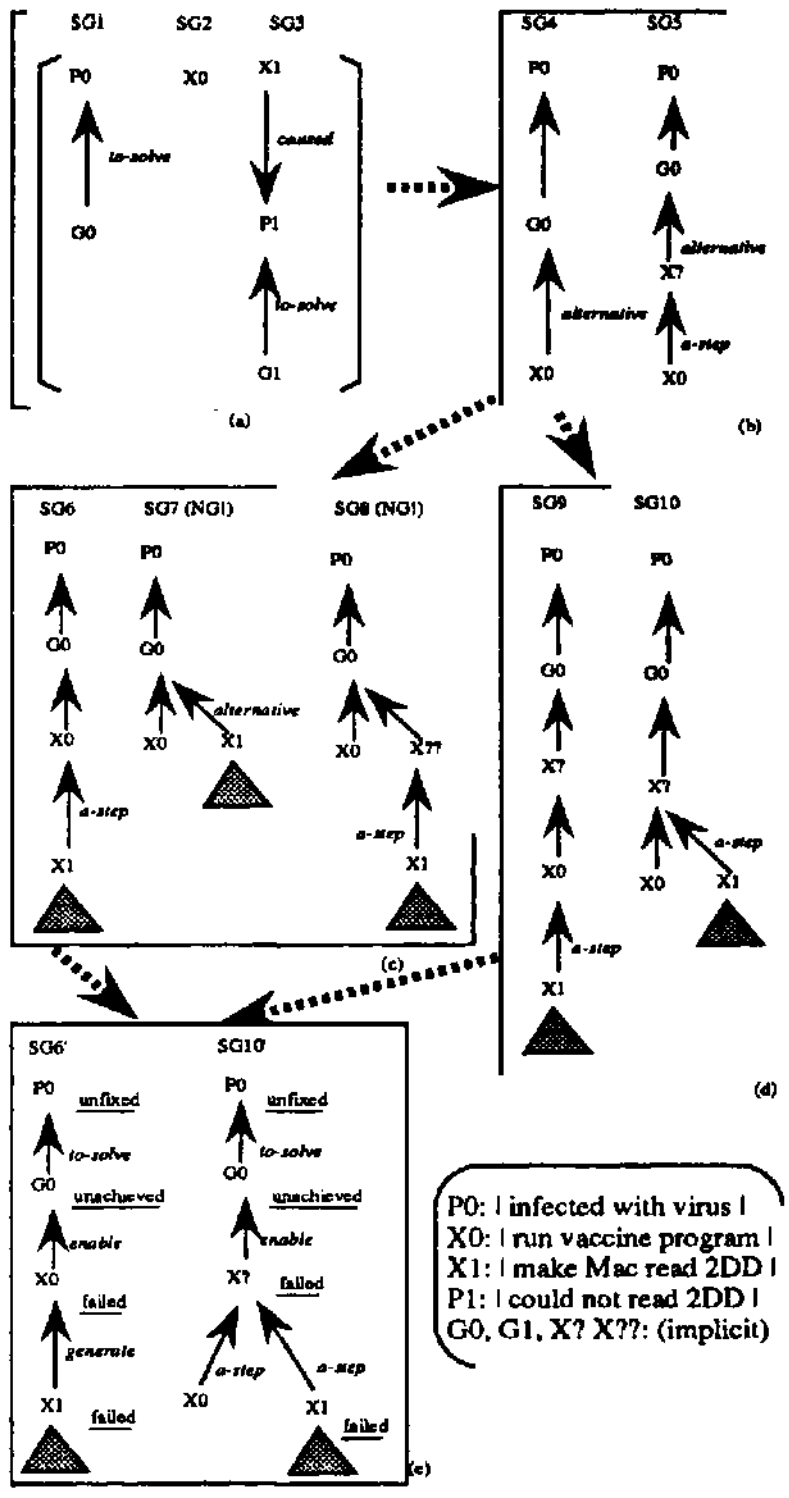

Figure 5: Trace of the Example.

5.4 Trace of the Example

We give a brief trace of the algorithm for the Example in Figure 5. The input set would be as shown in (a) [6], and the initial PS-graph is provided by SG1.

Main process: SG2 can be linked to G0 in SG1, and two PS-graphs, SG4 and SG5, result (shown in (b)). Therefore, the process tries to link the next element in

[6] X0 is provided by s2. As in the original message, s2 and s3 form the same sentence, X0 alone would also be introduced as an element of the input set.

the input, SG3, to both SG4 and SG5 in the next iteration. In SG4, GO and XO are possible linking points, but only one graph, SG6, result (shown in (c)). SG7 and SG8 are excluded by the heuristic based on the principle of alternative actions. On the other hand, in SG5, X? and XO are possible linking points, and SG9 and SG10 result (shown in (d)).

Post process: Three graphs, SG6, SG9 and SG10 are passed to the post process. By the procedure *reduce-graphs,* SG9 is identified as a redundant variant of SG10; therefore, it is thrown away. Two graphs, SG6[1] and SG10' are finally provided as possible PS-graphs (shown in (e)).

Note that the ambiguous relationship between s2 ("run vaccine") and s3 ("make Mac read a 2DD disk[1]'') is clearly captured by these two PS-graphs. Also, we know that the primary problem ("infected with virus[1]'') is still unfixed in either case from the PS-graphs.

6 Discussion

Clearly, the work reported here has a close relation to the so-called plan recognition. Therefore, it may be worth mentioning our view on the relation here. Roughly speaking, plan recognition in natural language understanding has long been studied from the perspective of *for* hearer, *understanding the speaker's utterance is recognizing the speaker's plan which generates the utterance,* and it is assumed that the hearer and the speaker have same plan library.

This assumption seems to be too strong to handle actual human dialogues/messages. For one thing, a plan which is valid for the speaker can be invalid for the hearer. Thus, we cannot assume it in actuality. One of the important works which addressed the issue would be [Pollack, 1990]. In the paper, Pollack views plans as complex mental attitudes and discusses how a speaker's invalid plans can be handled in dialogue understanding. Our work reported here was partly inspired by that work. However, as Woods points out in his comment[\Voods, 1990] to Pollack, she does not deal with partially specified plans. As shown in this paper, one can have an incomplete plan in a problem-solving activity. For example, one might plan to run a vaccine program in order to disinfect a computer virus without knowing the actual vaccine program. Our notion of the recovery of problem-solving activities can include such cases; incomplete plan or precondition failure in the plan recognition enterprise are handled themselves as one of the problematic situations.

Besides this, Woods made another important comment in his essay. He questions the role of surface expressions by introducing an example; "I want to get tenure, so I've rented a car". He says, "the conjunction *so* unambiguously signals the intended goal-achieving relationship without the necessity of determining the plan in order to infer it". This paper may provide an answer to this question, as we discussed particularly with the example.

7 Conclusions and Future Works

This paper proposed an efficient computational mechanism for recovering problem-solving activities from query messages without particular domain knowledge. This work, however, may be a first trial, and there remain several issues to be worked out.

From a computational viewpoint, we must first evaluate the actual performance of the mechanism through implementation and experiments with a larger set of unseen data. On the other hand, from a theoretical viewpoint, there are a number of harder issues. Particularly, our main concern at the moment is the formalization of human problem-solving (trouble-shooting) activities in terms of mental processes, as [Pollack, 1990] formalizes simple plans as complex mental attitudes.

Acknowledgments

1 would like to thank Jerry Hobbs for his valuable advice on this work, and Tsuneaki Kato for his useful comments on an earlier version of this paper. The work was done mainly while I was staying at CSLI, Stanford University. Thus, thanks also go to people at CSLI and NTT.

References

[Allen, 1984] Allen, J. Towards a general theory of action and time *Artificial Intelligence* 23, 123-154, 1984.

[Hammond, 1995] Hammond, K.R., Burke, R.C., Martin, C, and Lytinen, S. FAQ Finder: A Case-based Approach to Knowledge Navigation. In *Proceedings of the 11th Conference on Artificial Intelligence for Applications,* 80-86, Los Angels, CA, 1995.

[Hobbs, 1992] Hobbs, J.R., Appelt, D.E., Bear, J.S., Israel, D.J., and Tyson, W.M, FASTUS: A system for extracting information from natural-language text. Technical Note No.519, SRI International, Menlo Park, CA, 1992.

[Kitani, 1994] Kitani, T., Eriguchi, Y., and Hara, M. Pattern matching and discourse processing in information extraction from Japanese text. *Journal of Artificial Intelligence Research,* 2, 89-110, 1994.

[Krol, 1993] Krol, E. *The Whole Internet, User's Guide & Catalog.* O'R.eilly & Associates, Sebastopol, CA, 1993.

[Kuuo, 1987] Kuno, S. *Functional Syntax: Anaphora, Discourse and Empathy.* University of Chicago Press, OH, 1987.

[Masuoka, 1989] Masuoka, T., and Takubo, Y, *Kiso Nihongo Bunpou.* "Basic Japanese Grammar" (in Japanese), Kuroshio-Shuppan, Tokyo, Japan, 1989.

[Pollack, 1990] Pollack, M.E. Plans as complex mental attitudes, in Cohen, Morgan and Pollack (Eds.). *Intentions in Communication.* The MIT Press, Cambridge, MA, 1990.

[Woods, 1990] Woods, W.A. On plans and plan recognition: Comments on Pollack and on Kautz, in Cohen, Morgan and Pollack (Eds.). *Intentions in Communication.* The MIT Press, Cambridge, MA, 1990.