Case Based Reasoning with State Transition Mechanism for Problem-Solving in AI

Name: Arijit Chatterjee
Address: Vishwakarma Institute of Technology,
666, Upper Indira Nagar, Bibwewadi,
Pune, Maharashtra, India
E-Mail Address: techno_savvy81@yahoo.com

ABSTRACT
A method as an enhancement to Case Based Reasoning (CBR) systems, where each solution is treated as a case and a case database is used to search for its solution. However, unlike CBR systems, every Problem Solution Procedure is treated as a transition of an initial (present) problem state, in a problem domain into a final goal state in the solution domain called as a solution state, through a path of intermediate states in the problem and solution domain.
Instead of a case database, a path and pattern database is used that stores all problem and solution states and the paths to either of them including the intermediate states and the elements and methods responsible for state transition. Further on, instead of finding similarities with the case database, the path and pattern database is used to find the most feasible solution if one already exists, else a path is created using learning and the database is continuously enhanced to encompass paths to all tractable solutions, considering the present stock of resources and strategies to utilize those resources. The problem of identification and definition of problems is reduced considerably by utilizing causes and symptoms previously identified to be compared to those that have actually occurred and thus reducing the problem domain to a smaller problem space.

KEYWORDS
State, Path, Pattern, General Problem Solver, Case Based Reasoning, Learning.

I. INTRODUCTION:

There is need to develop a machine that has an ability to accept external knowledge found in sources like books, articles, databases e.t.c as knowledge elements and transform it into machine-simulated tacit knowledge in the form of intellectual activity support systems. They should assist during 4 steps of intellectual activity:
1. Observation
2. Producing propositions
3. Selection and verification of appropriate propositions
4. Memorizing (converting data to information to create new knowledge)
Since these machines have no human restrictions on knowledge volume, it is possible to input all existing knowledge into them that can then be used for adaptive learning. (Konstantin 1999).
The General Problem Solver (GPS) (Newell & Simon 1972) was a theory of human problem solving stated in the form of a simulation program (Ernst & Newell, 1969). GPS was intended to provide a core set of processes that could be used to solve a variety of different types of problems. The critical step in solving a problem with GPS is the definition of the problem space in terms of the goal to be achieved and the transformation rules.
A problem can be analysed into specific components. It consists of two situations, the present one which we will call the initial state, and the desired one, which we can call the goal state. The agent's task is to get from the initial state to the goal state by means of series of actions (Amarel 1968) that change the state. The problem is solved if such a series of actions has been found, to reach the goal.
Such a search process requires a series of actions, carefully selected from a repertoire of available actions to bring the present state closer to the goal (Heylighen 1988). Different actions will have different effects on the state. Some of these effects will bring the present state closer to the goal; others will rather push it farther away. To choose the best action at every moments of the problem-solving process, the agent needs some knowledge of the problem domain (Simon 1986). This knowledge will have the general form of a production rule: if the present state has certain properties, then perform a certain action. Such heuristic knowledge requires that the problem states be distinguished by their properties (Korf 1980).
A CBR system reasons by remembering previous decision problems (Gupta & Montazemi 1997): it uses their outcome to evaluate new decision problems (Kolodner & Mark 1992). The processes followed by a CBR system are (Riesbeck & Schank 1989, Sternberg 1977), to assist a decision maker (DM), previous case(s) that closely resemble the new decision problem (new case) is (are) retrieved. The solution of the previous case is then mapped as a solution for the new case. The mapped solution is adapted to account for the differences between a new case and a previous case.
Problem states will generally involve objects, which are the elements of the situation that are invariant under actions, and properties or predicates, which are the variable attributes of the objects. A problem state then can be formulated as a combination of proposition, where elementary propositions attribute a particular predicate to a particular object (Heylighen 1988). The different values of the predicates determine a set of possible propositions, and thus of possible states. Since states that differ only in one value of one predicate can be said to be "closer" together than states that differ in several values, the state set gets the structure of a space. Actions can now be represented as operators or transformations, which map one element of the state space onto another.
Every Problem Solution Procedure can thus be treated as transition of an initial problem state, in a problem domain into a

final goal state in the solution domain, called as a solution state, through a path of intermediate states.

The solution state at a later point of time may destabilise back into a problem state. Instead of a case database in a CBR System, a path and pattern database using GPS concepts can be used that stores all problem and solution states and the paths to either through the intermediate states and the elements interacting with these states to cause state changes though application of various methods.

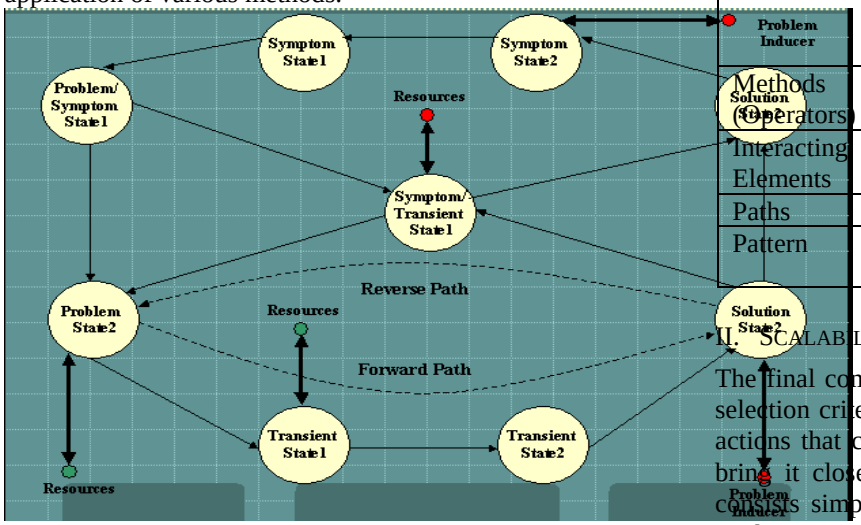

Fig 1. Problem and Solution Paths

The path from the problem state to the solution state is called a forward path and the path from a solution state to a problem state is called a reverse path. In the forward path, the interacting elements are called as resources, the methods are called as utilizations and the intermediate states are called as transient states. In the reverse path, the interacting elements are called as problem inducers the methods are called as the causes and the intermediate states are called as symptom states.

The collection of forward paths from one problem state to a solution state is called a solution pattern and the collection of reverse paths from one solution state to a problem state is called a problem pattern. Together they are called as patterns. Note that the intermediate states of one problem pattern can be problem state of another pattern and vice-versa. Further on time is an absolute factor and all pattern points are relative to it.

A set of factors that are essential for shaping any pattern, those define and shape the problem and its solution procedures into a path, which cumulatively form a pattern, can be called as pattern points.

A path and pattern database can be used to find the path to the most feasible solution; else, a path can be created using learning techniques. The database is continuously enhanced to encompass paths to all tractable solutions, considering the present stock of resources and strategies to utilize those resources. The problem of identification and definition of problems is reduced considerably by utilizing causes and symptoms previously identified to be compared to those that have actually occurred and thus reducing the problem domain to a smaller problem space.

Table 1. Table of Pattern Points

TIME		
Pattern Points	**Domains**	
	Problem	**Solution**
States	Problem State Intermediate state: Symptom Failure Transient	Solution State
Methods (Operators)	Causes	Solution Methods/ Strategies
Interacting Elements	Resources	Problem Inducers
Paths	Reverse	Forward
Pattern	Problem Pattern	Solution Pattern

II. Scalability of Solution Space: -

The final component we need to decide between actions is a selection criterion, which tells the agent which of the several actions that can be applied to a given state is most likely to bring it closer to the goal. In the simplest case, an action consists simply of moving to one of the neighboring states. Each state will then be associated with a certain value, which designates the degree to which it satisfies the goal. This value is called "fitness". (Korf 1980).

There are many solutions to problems, which are not ideal/final solutions or the final solution state has never been reached i.e. there exists the knowledge of the ideal solution state, but only a limited path from the initial problem state to a solution state comparatively closer to the solution state has been defined.

For solving such problems, whose ideal solutions are very difficult to reach or they haven't been identified, the person trying to solve the problem may have to scale down the solution space to find a solution state, which may not be ideal but gets the work done, depending upon its fitness value.

The threshold that separates the solution domain from the problem domain needs to be scaled up or down depending upon the degree of solution required and thus achieve scalability of solution space, with the ideal solution always lying in the final degree.

For problems with more than one solution, the solution with a higher threshold is called as a higher/enhanced solution and the solution with a comparatively lower threshold is called a lesser solution.

III. Path and Pattern Database: -

Here we create a relational database schema that can be utilised for storing and using the values of the pattern points of a problem-solution path along with associated time and costs.

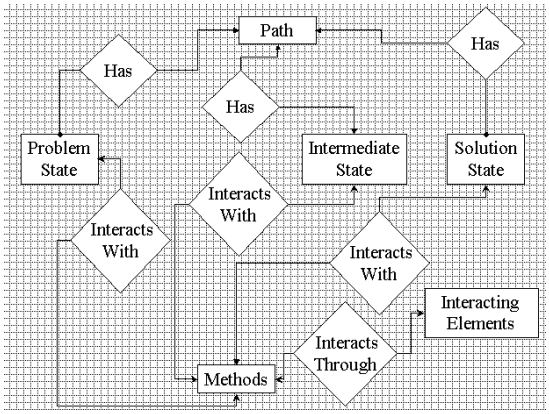

Fig 2. Entity Relationship Diagram

3.1 Problem Table: -

This table consists of all details about the problem state.

Problem_State(Problem_State_No*, Problem_State_Category, Problem_State_Proposition)

- Problem_State_No is a unique key identifying each problem state.
- Problem_State_Category specifies the category in which the problem lies in the Problem Domain.
- Problem_State_Proposition is the Problem Definition

3.2 Solution Table: -

This Table consists of all details about the Solution state.

Solution_State(Solution_State_No*, Solution_State_Proposition)

- Solution_State_No is a unique key identifying each solution state.
- Solution_State_Proposition is the degree of solution required which is dependent upon the solution threshold set.

The solution state proposition is the degree of solution required and the problem state proposition is the problem definition.

3.3 Intermediate State Table: -

Intermediate states (symptom states and transient states) are the states when Strategies (tasks for project based approach and exercises for exercise based approach) and causes are applied to resources and problem inducers for their utilizations and causes respectively.

Intermediate_State(Intermediate_State_No*,Intermediate_State_Category, Intermediate_State_Proposition, Intermediate_State_Type)

- Intermediate_State_No is a unique key identifying each intermediate state.
- Intermediate_State_Category specifies the problem category in which the state lies.
- Intermediate_State_Proposition is the description of the intermediate state
- Intermediate_State_Type specifies the type of Intermediate state i.e. Symptom State or Transient State)

3.4 Interacting Elements Table: -

This Table consists of the details of all Interacting Elements used in the problem domain

Interacting_Elements(Interacting_Elements_No*,Interacting_Elements_Name, Interacting_Elements_Type, Interacting_Elements_Cost, Interacting_Elements_Time)

- Interacting_Elements_No is a unique key identifying each Interacting Element in the problem domain
- Interacting_Elements_Name specifies the name of each interacting element.
- Interacting_Elements_Type specifies the type of interacting element i.e. Problem Inducer or Resource.
- Interacting_Elements_Cost specifies the cost that needs to be incurred to eliminate a problem inducer or utilize a resource.
- Interacting_Elements_Time specifies the total time required to eliminate a problem Inducer or utilize a resource.

3.5 Method Table: -

This Table consists of the details of all Methods used in the problem domain

Method (Method_No*, Method_Name, Method_Type, Method_Cost, Method_Time)

- Method_No is a unique key identifying each Method in the problem domain.
- Method_Name specifies the name of each method.
- Method_Type specifies the type of each method i.e. Cause or Strategy.
- Method_Cost specifies the cost that needs to be incurred to apply a strategy or eliminate/reduce the effect of a problem cause.
- Method_Time specifies the total time required to apply a strategy or eliminate/ reduce the effect of a problem cause.

This table gives the details of all methods with all associated interacting elements of each method.

Method_Interacting_Elements(Method_No*, Interacting_Elements_No*, Method_Details)

- Method_Details specifies the details of how an interacting element interacts with a state through the given method.

This table gives all methods used for each state:

Method_State (Method_No*, Method_State_No*)

- Method_State_No gives the problem state no, solution state no or intermediate state no with which the method is interacting.

3.6 Paths Table: -

This Table consists of all problem (reverse) and solution (forward) paths with all states achieved including problem state, solution state and intermediate states.

Path (Path_No*, Problem_State_No*, Solution_State_No*, Path_Type, Path_Innovated)
Path_No is a unique key identifying each path in the problem domain.
Path_Type specifies the type of path i.e. reverse or forward.
Path_Innovated specifies whether the path is an innovated path or not (Takes values either Yes or No).
Problem_State_No and Solution_State_No specifies the initial and final states of the path depending upon its Path_type, i.e. Problem State is initial state and Solution State is final state if the path is of forward type and vice-versa if path is of reverse type.

This table gives all the intermediate states in each path.
Path_Intermediate_State (Path_No*, Intermediate_State_No*)

4 CREATION OF INNOVATED PATHS & POPULATING DATABASE: -

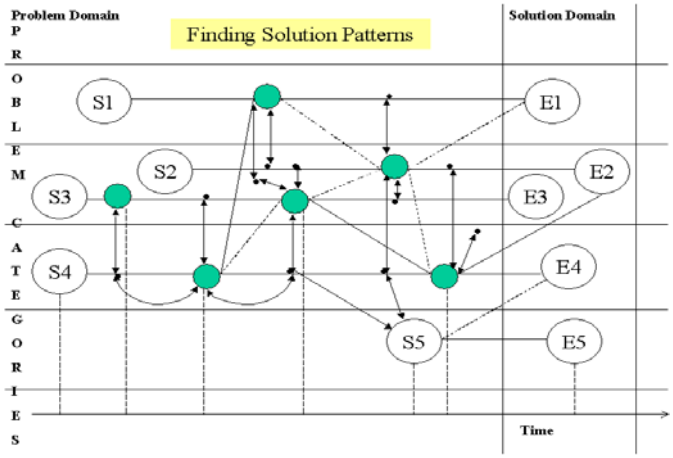

Fig 4. Patterns and Innovated Paths

In above diagram, the uncolored circles are the problem and the solution states and the colored circles are the intermediate steps that occur due to the application of certain methods to utilize certain resources, depicted by the double-direction arrows.
From the figure, we can identify two types of paths via:

- The lined paths represent the solution path that needs to be taken for problem solution through the application of methods used in well-established problem solution techniques. This is called as a solution pattern which is a collection of all solution paths from a given problem state to a solution state.
- The dotted path represent the path that might have been taken as innovation during the solution of some problem. This termed as the innovated solution path.

The same is applicable for identifying problem paths and innovated problem paths.
We can populate the path and pattern database, with all known and identified problem/solution paths for each problem domain.

We then enhance the database with innovated paths, which are paths that can be logically derived from the combination of paths and patterns in the database. Using the mapping process of CBR systems to previously identified paths, innovated paths to all possible solutions not yet identified, become apparent when the well-established paths are combined together.

4.1 Steps to generate innovated solution paths according to steps of Intellectual Activity: -

1. Categorize all problem situations (Observe)
2. Identify all the pattern points in the problem category (Produce, Select & Verify Propositions).
3. Compare solution paths using the following algorithm to create innovated paths (Memorize):
    1. Start.
    2. For each problem state, identify Strategies applied on it from the solution paths of the problem category.
    3. For every Solution method applied, identify the transient state achieved by the problem state.
    4. If the state is a solution state, with a path different from the solution path being traversed, store the pattern in the path table of the database as an innovated solution path and goto step 8, else goto step 5.
    5. Find the Strategies applied on the transient state from the solution paths of the problem category.
    6. For each Solution method applied on the transient state, check if the state is a solution state from the solution states of the problem category.
    7. If the state is a solution state, with a path different from the solution path being traversed, store the pattern in the path table of the database as an innovated solution path and goto step 8, else goto step 5.
    8. Perform step 2 until all Strategies applied to the transient state is processed.
    9. Perform step 1 until all Strategies applied to the problem state is processed.
    10. Stop.

The same steps can be applied by replacing problem state with solution state, solution state with problem state, strategies with causes and resources with problem inducers to find innovated problem paths.

4.2 Improvement Criteria: -

The efficiency of problem solving is strongly determined by the way the problem is analyzed into separate components: objects, predicates, state space, operators, and selection criteria. This is called the problem representation.
The factors on which the efficiency of the generation of Innovated Paths depend on are as follows:

1. Number of Non-Innovated Paths identified in the database.
2. Number of problem and solution states Database.
3. Degree of categorization of problems.

5 ANALYSING EXISTING & INNOVATED PATHS FOR LEARNING TO CREATE NEW PATHS:

In simple control problems, the solution is trivial e.g., the thermostat is an agent whose goal is to reach or maintain a specific temperature. The initial state is the present temperature. The action consists in either heating to increase the temperature or cooling to decrease it. The decision which of these two possible actions to apply is trivial: if the initial temperature is lower than the goal temperature, then heat; if it is higher, then cool; if it is the same, then do nothing. Such problems are solved by a deterministic algorithm: at every decision point there is only one correct choice. This choice is guaranteed to bring the agent to the desired solution.

The situations we usually call "problems" have a more complex structure. There is choice of possible actions, none of which is obviously the right one. The most general approach to tackle such processes is generate and test: apply an action to generate a new state, then test whether the state is the goal state; if it is not, then repeat the procedure. This principle is equivalent to trial-and-error, or to evolution's variation and selection. The repeated application of 'generate and test ' determines a search process, exploring different possibilities until the goal is found. Searches can be short or long depending on the complexity of the problem and the efficiency of the agent's problem-solving strategy or heuristic. Searches may in fact be infinitely long: even if a solution exists, there is no guarantee that the agent will find it in a finite time.

If there does not exist a path from a given problem state to a desired solution state, learning through analyzing the existing paths in the database can be applied to populate the database with new paths as enhancements to the all the paths identified till date.

This can be performed as follows: -

- Identify all pattern points in the database
- Apply a breadth first or depth first search for identifying paths to the possible goal/solution states (Dean, et al 1999) from all the identified initial/problem states by applying the methods one by one to the initial problem.
- Store the new path in the database, and perform steps 3 to 5 until all states in the problem domain have been covered.

The same can be applied to identify paths from solution state as the initial state to problem state as the goal state in order to create new problem paths

Consider the search space shown below:

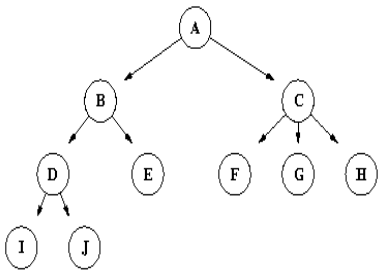

Fig 5.Search Space

Let us suppose that the state labelled G is a goal state in this space, and that, as shown, no operators apply to the states I, J, E, F and H.

A program will start with the initial state A, and try to find the goal by applying operators to that state, and then to the states B and/or C that result, and so forth. The idea is to find a goal state, and often one is also interested in finding the goal state as fast as possible, or in finding a solution that requires the minimum number of steps, or satisfies some other requirements.

One approach to deciding which states that operators will be applied to is called "Breadth-First Search". In a breadth-first search, all of the states at one level of the tree are considered before any of the states at the next lower level. Therefore, the states would be applied in the order indicated by the dotted blue line:

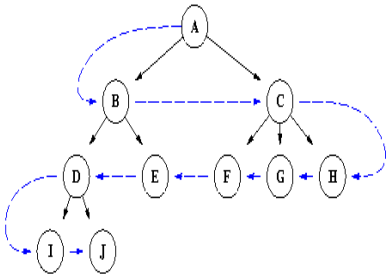

Fig 6. Breadth First Search to search Space

Another approach to deciding which states that operators will be applied to is called "Depth-First Search". In a depth-first search, after operators are applied to a state, one of the resultant states is considered next. Therefore, the order in which the states of the simple example space will be considered is:

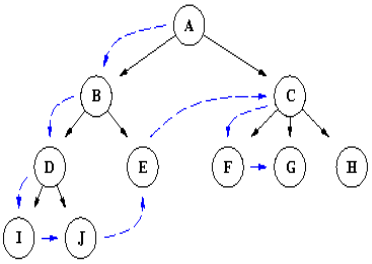

Fig 7. Depth First Search to search Space

If a node is a failure node or there are no applicable operators, the next node to be considered might be in the level above that of the current node e.g. Node J, is a failure node, and so the next node to consider is the remaining node at the level above.

In addition, in this diagram we have assumed that when a state is considered, all of the applicable operators are applied to the state. This isn't always necessary e.g., one could apply only one of the possible operators to each node, and then one of the possible operators to the result and so on.

Breadth-First and Depth-First search are sometimes called "blind" or "knowledge-free" search techniques because they incorporate no specific information about the problem domain except that it can be described as a search

These two search techniques have useful properties of their own, and in many cases, their simplicity makes them more practical than fancier approaches.

In particular, Breadth-First search can be proved to have the following properties:

1. If a solution exists in the search space, Breadth-First search will eventually find it.
2. Breadth-First search will find the shortest possible solution, measured in terms of the number of operator applications.

Suppose that there are a number of solutions (perhaps just one) in a search space. Suppose further that the one at the lowest level (remember that each level corresponds to one operator application) is at some level N. From the way, that Breadth-First search is defined; we know that it will consider all of the states at level N before it considers any state at a deeper level. Therefore, if the minimal (or only) solution is at some level N, Breadth-First search will find it.

Depth-First search can't be proved to satisfy either of the above two properties. In particular, if the search space is infinite, Depth-First search might head down one branch of the search tree and never return, even though a solution might exist only a few levels down another branch.

On the other hand, note that for Breadth-First search to find the minimal solution at level N, it must consider at least 2^N search states. This might take a while, if N is very large at all. Suppose that in some search domain, there are very many solutions, but all of them are at least 10 levels into the space. Breadth-First search will spend lots of time exploring all of the states at all of the levels below that, while Depth-First search will dive right down to level 10 and presumably find a solution there.

Another case where Depth-First search performs well is when most of the space are failure states, or states where no operators can be applied. In such a search space, it is often best to try sequences of moves until they are known not to work, and then to "backtrack" back to the last legal state.

6 CREATION OF A DSS FROM THE DATABASE: -

Here we use the Database for Problem Identification, Definition and efficient Resource Planning and creation of the Main as well as Contingency Resolution Procedures for resource utilization, via:

.

1. Accept statements describing problematic situations the person having a problem i.e. user, is facing.
2. Parse the sentences to pick up nouns and verbs and put them in a wordlist.
3. Create a dictionary of words for every problem category identified and apply a category wise search of the wordlist from the parsed sentences, to match with the words in the dictionary, to identify the problem categories. This can be done through various ways via:
    a. Advanced query expansion and disambiguation tools, including linguistic stemming and thesaurus expansion.
    b. Custom thesaurus creation.
    c. Natural language query input.
    d. Automatic highlighting of search terms and linguistic and thesaurus equivalents.
    e. Combined metadata and full text search:
    f. Advanced query navigation.
    g. Rich query language, including query operators, proprietary fuzzy search technology.
4. Identify the problem states within the problem categories and their corresponding causes.
5. Display list of symptom states that have a reverse path existing between the symptom state and the identified problem states.
6. Ask user to identify the symptoms that have actually occurred.
7. Depending upon the symptoms selected, list the possible causes of these symptoms.
8. Ask user to identify the causes that have actually occurred.
9. From the identified causes, identify the symptoms that can be generated due to these causes from the selected symptoms.
10. Depending on the identified symptoms, reduce the number of possible problem states that can be transitioned from the identified symptoms.
11. List the solution states that can be reached from the most possible problem states through traversing the solution paths existing between the two end states and the corresponding cost and time to reach the solution.
12. Based on the most possible problems and their possible solutions, ask user to identify the problem state, the solution of which can achieve the results desired by him.
13. Hold a feasibility analysis on the solution paths between the selected problem state and the selected solution states, based on the total cost and time involved.
14. Identify the most feasible path along with the methods applied, and resources used in the path.
15. Identify all resources that had been used in the path identified in step 11, to perform resource planning.
16. Identify all other paths in the solution pattern having the most feasible path, to be used as contingency resolution procedures.

In the above method, paths and patterns are analysed to identify all possible solutions to a given problem along with the cost and time associated with it. It further finds out possible enhancements to a given solution and leaves open the scope of learning from past-solved cases of problems to find new solution procedures.

7 MONITORING AGAINST FAILURES AND CONTINGENCIES-

The paths to be used for solution can be used as a process trajectory, and can be monitored and controlled during implementation by systems that continuously monitor deviations from original path by checking the solution procedure during every state change.

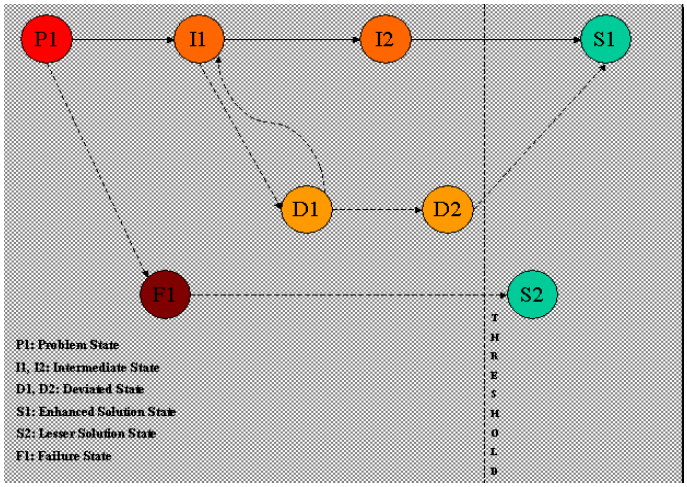

Fig 8. Contingency Situation Resolution

In case a contingency situation arises such that there is a deviation from a transient state to a deviated state, two resolution methods can be applied, depending upon the cost-time constraints, via: -

- Bring back to previous state.

Through application of learning to create new paths, it is possible to realize a path that can be followed to transform a deviated state as an initial state into the previous state as the goal state.

- Choose a different path from deviated state.

Sometimes it is not feasible to transform back into the previous state, if another path exists towards the solution from the deviated state, if it takes less time or cost than for transforming back into previous state.

Failure states are those states, which upon their arrival ensure that any number of methods/operators, applied to that state or series of its subsequent states, can never achieve a goal state.

A failure state can be identified when: -

1. If the application of any method on a state doesn't cause a change of state.
2. If the path continuously keeps coming back to a state already traversed more than once.
3. If the application of methods/operators, doesn't bring in a goal/objective state even after a huge number of iterations. (Mostly for NP-Complete type of problems)

Failure states of the first category are quite easy to identify. The failure states of the second and third category might require a large number of iterations to be identified, and thus impossible to identify with the present limitations of time and resource i.e. their existence makes the solution intractable.

If there a Failure State is reached during solution, loss minimization needs to be performed by changing the failure state into a lesser solution state feasible to the user.

REFERENCES

Amarel S. (1968): 'On Representations of Problems of Reasoning about Actions', in : Machine Intelligence 3, D. Michie (ed.) (American Elsevier, New York).

A.R. Montazemi and L. Chan (1990), An analysis of the structure of expert knowledge, European Journal of Operational Research 45 (1990) pp 275-292.

A.R. Montazemi and K.M. Gupta, "An adaptive agent for case description in diagnostic CBR system", Journal of Computers in Industry, 29(3) (1996) pp 209-224.

C.K. Riesbeck and R.C. Schank (1989), "Inside Case-Based Reasoning", (Lawrence Erlbaum, Hillside, N J, 1989).

Dean, et al (1999), "Problem Solving and Search", Cognitive Science 108b Lecture Notes, Sections 4.1, 4.2.

Heylighen F. (1988): "Formulating the Problem of Problem-Formulation", in: Cybernetics and Systems '88, Trappl R. (ed.), (Kluwer Academic Publishers, Dordrecht), p. 949-957

Heylighen F. (1990): Representation and Change. A Metarepresentational Framework for the Foundations of Physical and Cognitive Science, (Communication & Cognition, Gent), 200 p.

Kalyan Moy Gupta, Ali Reza Montazemi (1997), "A connectionist approach for similarity assessment in case-based reasoning systems", Decision Support Systems 19 (1997) pp 237-253

Konstantin M Golubev, Tatiana A Golubeva (1999), "Intellectual activity, knowledge, information, data: An attempt to define it in an applicable way"(online), Knowledge Management Discussion Paper, General Knowledge Machine Research Group, Kiev, Ukraine.

Korf R.E. (1980): 'Toward a Model of Representation Changes', Artificial Intelligence 14, p. 41.

Newell A. & Simon H.A. (1972): "Human Problem Solving", (Prentice-Hall, Englewood Cliffs)

Simon H.A. (1986) et al.: "Decision Making and Problem Solving"

R.J. Sternberg (1977), "Component processes in analogical reasoning", Psychological Review 84(4) (1977) pp 353-378.