Proceedings of the 1999 IEEE
International Symposium on Assembly and Task Planning
Porto, Portugal - July 1999

Failure Recovery Planning in Assembly based on Acquired Experience: Learning by Analogy

L. Seabra Lopes
Departamento de Electrónica e Telecomunicações
Universidade de Aveiro – 3800 Aveiro - Portugal

Abstract

For complex tasks in flexible manufacturing as well as service applications, robots need to reason about the tasks and the environment in order to make decisions. This is a topic that is far from receiving from the robotics community the due attention. This paper presents a method for recovering from execution failures based on analogies with previous failure recovery episodes. The basic principles that explain the success of a failure recovery strategy are extracted based on several deductive as well as inductive transformations. In recovery planning based on these learned principles, the inverse transformations are applied.

1. Introduction

Robots are the most flexible elements currently used in manufacturing systems. In the assembly area, if other flexibility elements, such as sophisticated end-effectors and flexible, especially modular fixtures, are introduced, then robots will potentially be able to perform a greater variety of tasks with a higher complexity [12,13]. Robots are, therefore, a central component in flexible manufacturing and assembly systems (FMS/FAS). Note, however, that, in this context, the keyword *flexibility* is generally understood as *the ability to cope with change*. This implies that the robot should be able to choose which actions to perform in each situation. This is a particularly important topic, especially in what concerns failure detection, diagnosis and recovery planning.

In the service sector, where robots are expected to work in unstructured environments and to act, within certain limits, independently [8], the problem of decision-making arises again.

My work in recent years has been concerned with the development of robot architectures that support decision-making at the task level [9]. The long-term goal is to build robots that can be instructed in the domain of concepts of the human user and, preferably, in natural language. This subject has been receiving surprisingly little attention in the major robotics journals and conferences. An exception is described in [2].

The approach for the task level part of the robot architecture is assumedly a symbolic and logic-based approach. The classical problem of symbol grounding is addressed via learning and via human-robot interaction. Previous work has focused on learning failure diagnosis and recovery knowledge to support the execution of assembly tasks [1,9,11].

In the present paper, I take up the part of the work concerned with failure recovery planning based on experience of similar failures previously encountered and successfully recovered. An initial contribution in this direction was given in [10].

The major problem to face in the domain of failure recovery is concerned with the search complexity involved in real-world planning. A robot has limited time to plan and recover from a situation.

Even in very circumscribed domains, planning complexity prevents classical artificial intelligence planners from finding solutions to problems that require long action sequences. In real-world domains, of course, the number of action alternatives is much bigger and complexity becomes a problem, even when only a short sequence of actions is to be determined. For instance, in the assembly domain, a sophisticated hand or a modular fixture offer a variety of possibilities for solving a variety of tasks. However, finding the right sequence of arm moves and hand and fixture configurations for solving some unexpected problem is not easy.

Avoiding planning complexity by using knowledge about solutions to similar, previously encountered planning problems is a way out that has been attracting increasing attention from researchers in various domains. This learning approach involves the recognition of the basic principles underlying those solutions and the application of those principles to new situations. In the failure recovery domain, the descriptions of failure categories and operator schemata make up a domain theory that must be taken into account when explaining (or understanding) the recovery strategy that was applied in a given situation.

Traditional feature-based learning algorithms are not suited for addressing this problem. On one hand they are limited to learning a fixed set of classes. This is not desirable because there isn't a fixed set of failure situations and failure recovery strategies to apply. On the other hand, feature-based algorithms lack the ability for explanation.

The approach followed in this paper takes some inspiration from the literature on Case-Based Reasoning (CBR) [4] and Explanation-Based Learning (EBL) [7,3].

However, some important differences in the approach exist, as will be pointed out.

In a first step, the description of the failure recovery episode will be deductively generalised. Then, a series of transformations (namely abstraction, feature extraction and clustering of repeated plan patterns) will remove irrelevant details. The new representation obtained in this way, called a *failure recovery schema*, summarises the key aspects of the solution that was applied and are potentially relevant for guiding failure recovery planning in a variety of related situations. All learning and planning is implemented in Prolog language.

In section 2, the domain theory and the representation of failure recovery episodes will be presented. Section 3 concentrates on learning of failure recovery schemata. Section 4 describes how new plans are generated by adapting old solutions. An example is given in section 5 and the conclusions will be presented in section 6.

2. Domain Theory and Adopted Representations

As already mentioned, the adopted approach is a symbolic and logic-based approach. The world model is composed of a *situation description* (a set of atomic formulas, called *situation descriptors*, that describe various aspects of the situation of the robot's world in a given moment) and the *world information* (a set of formulas describing permanent properties of the world). The world can change through the actions of the robot, modelled as *operator schemata*, and through unexpected events, modelled as *failure categories*. All these representations are symbolic in nature, although they may include numerical properties of the physical world.

A failure recovery episode consists of the execution of a sequence of operations in order to recover from a failure situation. In general, learning can occur based both on successful and unsuccessful failure recovery episodes. Successful recovery happens, of course, when the planned recovery operations are completely executed, allowing to resume execution of the nominal plan. If some exception, unrelated to the initial failure, causes a new failure which is handled recursively, and then the initial recovery strategy is resumed and completed, recovery is also considered successful. Unsuccessful recovery may happen due either to the application of an incorrect recovery strategy, given the initial failure diagnosis, or to errors in the initial diagnosis itself. The first case can usually be avoided by verifying the effects of the recovery strategy, according to the applied operator schemata, before execution. In the second case, learning is more difficult because it involves criticising and reformulating the used diagnostic model. In this paper, learning will be attempted only from successful failure recovery episodes.

The adopted representation of robot actions has been presented in previous publications [9,10]. Robot actions are operator applications. The functionality of an operator is specified in an *operator schema* in terms of lists of effects (the add list and the delete list), a list of pre-conditions that are not deleted (keep list) and a list of permanent properties that are relevant (information list). An operator schema is identified by the operator template. For instance, `pickup_object(R,T,Obj)` is the template of the operator schema that describes de effects of robot `R`, with tool `T` attached, picking an object `Obj` from the worktable. A control function, specifying how operator applications are actually executed in the physical world, must be associated to each operator.

Execution failures are described by *failure categories*, whose representation is somewhat similar to that of operator schemata. A failure category is a tuple (*FT,FOp,DL,AL*), where

- *FT* is the failure category template,
- *FOp* is the template of the failed operation, and
- *DL* and *AL* are the delete and add lists, specifying the failure effects.

The specification of failure effects assumes that none of the effects of the failed operation were actually achieved.

Defective parts and assemblies are among the external exceptions that may lead to execution failures. As an example, consider that when mating a part to a partially assembled product, the mate fails and it turns out that the part was defective. This is an instance of the failure category presented in fig. 1.

```
failure_category(
  defective_part(Obj,Type),
  assemble(R,T,Obj,Comp,Type,Prd,Fx),
  [ ], % delete list
  [ defective(Obj) ] % add list
).
```

Fig. 1 — A failure category

The description of a failure actually occurred in the physical scenario is an instanciation of a failure category and is identified by the instanciated failure template.

Finally, the representation of a failure recovery episode must identify the failure, the failed operation and the applied recovery strategy. A failure recovery episode is, therefore, a tuple (*FOp,FD,RPlan*), where

- *FOp* is the template of the failed operation
- *FD* is the failure description template, i.e. an instanciation of the corresponding failure category template
- *RPlan* is the applied recovery plan, i.e. a sequence of operator applications

The first time the robot faces an instance of a given failure category, the recovery strategy can be determined in two different ways: 1) there is some knowledge, heuristics or previous cases that, being tailored for or originating in different situations, still help a powerful planner to calculate the needed recovery strategy in reasonable time; or 2) a human teacher provides it.

I have been using the Cranfield Benchmark as a source of examples. Fig. 2 illustrates an instance of the failure category described in fig. 1. In this case, when assembling the crossbar (`cb1`), the operation failed due to a defect in the part.

Fig. 3 presents de complete failure recovery episode. The applied recovery strategy consists, basically, of discarding the defective part, obtaining a new one of the

same type (type is cb in this case) and assembling it. (The representations reproduced in this paper are exactly those used in the implemented system; some details may seem obscure but, I hope, the reader will be able to capture the basic ideas).

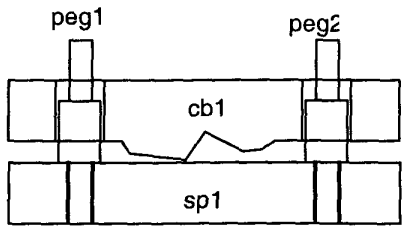

Fig. 2 — Mate failure due to defective crossbar

```
[ assemble(sony,gp_2,obj7,cb1,cb,cranf,fx),
  defective_part(obj7,cb1),
  [ putdown(sony,gp2,obj7),
    feed_part(fd_cb,obj88,cb),
    pick_from_feeder(sony,gp2,obj8,cb,fd_cb),
    assemble(sony,gp2,obj8,cb1,cb,cranf,fx)
] ]
```

Fig. 3 — Failure recovery episode (see figs. 1 and 2)

The descriptions in figs. 1 and 3 are written in Prolog. The reader should note, in particular, that initially capitalised arguments are variables while the others are constants (concrete entities).

3. Learning a Failure Recovery Schema

The representation of the basic principles that explain the success of a failure recovery solution applied in a given situation is called a *failure recovery schema*. It can be used for guiding recovery in future instances of the same failure category. In this section, the implemented method for building failure recovery schemata is outlined. All this is discussed at length in [9].

3.1. Deductive Generalisation

The first step is the generalisation of the description of the failure recovery episode. The goal is to produce a structure similar to the description of the failure recovery episode, but in which all constants are replaced by variables. The variables in the generalised description are not arbitrary. Each constant, wherever it appears playing the same role in the concrete episode description, is consistently replaced by the same variable. Therefore, generalisation can only be done based on a deep understanding (in the sense of deductive explanation) of the failure recovery episode, taking into account the domain theory. The specific generalisation method that is applied finds its roots in explanation-based learning (EBL) [3,7] and goes through the following steps:

- *Variabilisation*: replace each occurrence of a constant by a different variable.
- *Explanation*: use the domain knowledge to explain the failure effects and failure recovery plan and, at the same time, build a partial description of the failure situation, as required by the failure episode, and gather all permanent information facts that are relevant.
- *Generalisation*: unify the variables that represent the same entities in the explanation of the concrete episode; generalisation is done in parallel and guided by the explanation of the concrete episode.
- *Criticism*: establish additional bindings on variables.

As a side effect of explaining and generalising a failure recovery episode `FRE = [FOp,FD,RPlan]`, three additional items of information are produced:

- the generalisation of the permanent information facts that were consulted during explanation
- the generalisation of the failure situation, i.e. the situation of the world immediately after failure occurrence
- the generalisation of the description of the goal situation (post-recovery situation)

This information will be important to extract certain distinctive features of the failure episode (section 3.3).

3.2 Abstraction

Failures seldom repeat with exactly the same characteristics. In contrast, with only minor adjustments, the same basic solution often applies to a large variety of failures. Applying abstraction is a common approach to reduce the level of detail in a representation. Note that, unlike the common wisdom, abstracting a representation does not produce a "more general" representation. In fact, while abstraction reduces de level of detail in a representation, generalisation enlarges the domain of entities to which the representation applies [5].

The particular approach that was followed in this research is to use a taxonomy of operators as the reference structure for abstraction (see fig. 4).

```
pick(Obj)
 • pickup_object(R,T,Obj)
 • pickup_part(R,T,Obj,Part)
 • pick_from_pallet(R,T,Obj,Part,Pal,Pal_sys)
 • pick_from_feeder(R,T,Obj,Part,Fd)

place(Obj)
 • putdown_object(R,T,Obj)
 • place_in_pallet(R,T,Obj,Part,Pal,Pal_sys)

assemble(Obj,Comp)
 • assemble(R,T,Obj,Comp,Part,Prod,Fix)

disassemble(Obj,Comp)
 • disassemble(R,T,Obj,Comp,Part,Prod,Fix)

nil
 • feed_pallet(Pal,Pal_sys)
 • putaway_pallet(Pal,Pal_sys)
 • get_tool(R,T,TP)
 • store_tool(R,T,TP)
 • set_fixture_for_product(Fix,Prod)
```

Fig. 4 — Taxonomy of operators

Each operation in the recovery plan is abstracted according to the taxonomy. The class of operations

represented as `nil` contains operations that don't have a corresponding abstract operation and, therefore, disappear when a plan is abstracted.

The abstract operators also have their functionality defined by operator schemata in terms of abstract situation description predicates (this is one of the differences with respect to the initially published version of the system [10]). A taxonomy of situation description predicates is also necessary in order to be aple to prove abstract plans. Fig. 5 shows a fragment of that taxonomy.

```
available(Obj)
• part_available_in_feeder(Obj,Fd)
• part_in_pallet(Obj,Pal)
• object_on_table(Obj)
.....
```

Fig. 5 — Abstraction of situation description predicates

In this way, the `pick(X)` abstract operator can be defined by the delete list `[available(X), robot_free]` and the add list `[holding(X)]`.

3.3. Feature Extraction

When a failure is detected, a failure recovery strategy will be determined (if possible) by retrieving and adapting a previous solution, embodied in a failure recovery schema. The typicality of the calculated strategy as an instance of that generic solution is assessed by evaluating a set of features.

Features are traditionally used to describe examples in inductive learning and pattern recognition and to index and retrieve cases in case-based reasoning. Extracted features make explicit the relationships that would otherwise be implicit in the input. A feature transformation phase is often applied to the raw data before problem solving [11].

In the present work, features are constructed for all entities appearing in the abstract plan but not appearing in the indexing key. The *indexing key* is the piece of information that must be provided in order to locate in memory the failure recovery schema most appropriate to help in a new situation. In the implementation that was developed *the indexing key is composed of the failed operation template plus the failure description template.* A feature of an entity is a relationship between that entity and another entity appearing in the indexing key. As in other learning approaches, feature extraction strategies must be defined by the human operator. New feature definitions can be added to the system at any time without affecting what was previously learned. These features will be used to choose the best instanciations of the plan skeleton during plan adaptation. Examples of features in the assembly domain are given in fig. 6.

From the failure recovery episode proposed as example in figs 1, 2 and 3, we would obtain, after generalisation, abstraction and feature extraction, the failure recovery schema presented in fig. 7.

`initially_in_contact(P1,P2)`

— *the parts* `P1` *and* `P2` *were in contact in the initial situation (the failure situation, before recovery)*

`initially_assembled(C)`

— *component* `C` *was assembled in the initial situation*

`same_type_as(P1,P2)`

— *the type of part* `P1` *is the same as part* `P2`

`in_contact_with(C1,C2)`

— *the components* `C1` *and* `C2` *are in contact when the product is assembled*

`assemble_before(C1,C2)`

— *component* `C1` *must be assembled before* `C2`

`assemble_after(C1,C2)`

— *component* `C1` *must be assembled after* `C2`

Fig. 6 — Features in failure recovery schemata

```
[ assemble(R,T,DefectObj,Comp,Type,Prod,Fix),
  defective_part(DefectObj,Comp),
  [ [ place(DefectiveObj), [] ],
    [ pick(NewObj),
      [same_type_as(NewObj,DefectObj)] ],
    [ assemble(NewObj,Comp),
      [same_type_as(NewObj,DefectObj)]]
] ]
```

Fig. 7 — A failure recovery schema (see figs. 1-3)

In this case, only the second and third steps of the abstract recovery plan were documented with features (displayed in italic). In both cases, only a `same_type_as` feature was extracted. It means that the new part (`NewObj`), used to replace the defective part, must be of the same type of the replaced part.

Note that, while in the original episode the part that is used instead of the defective part is picked from a feeder (fig. 3), the abstract recovery plan allows the new part to be fetched in other places, for instance in a pallet or in the worktable. This kind of flexibility is not available when strictly following the generalised recovery plan.

3.4. Clustering of Repeated Plan Patterns

The recognition of patterns in the abstract plan can also contribute to a more generic description of the applied solution. For instance, in the example to be presented in section 7, disassembling and putting components on the table, or picking up and assembling them, are two patterns that occur repeatedly. It often happens that the solutions to two different problems only differ in the number of repetitions of some action pattern. In the learning system that was developed, repetitions of action patterns in a recovery plan are recognised and clustered.

Two sequences of operations are considered to belong to the same pattern if the following conditions hold:

- the names and order of the operations are the same;

- the variables representing entities mentioned in the indexing key (failed operation and failure description templates) are the same and play the same role; and
- to each variable playing a given role in a sequence, corresponds another variable playing that same role in the other sequence.

In the plan pattern recognition algorithm (given in [9]), priority is given to finding repetitions of short patterns. For increasing values of pattern length, K, the possibility of the plan containing two or more occurrences of a pattern of that length, starting in the first operation in the plan, is investigated. If that is not the case, the same search is conducted starting in the second operation of the plan, and so on. Of course, after finding the first pattern repetition, the algorithm looks for other patterns until the entire plan is searched. The algorithm does not take into account pattern repetitions inside longer patterns (this is a difference with respect to the first version of the method [10]). When generating the failure recovery schema, a sequence of repetitions of the same action pattern is substituted by a so-called *cluster*, i.e. an instance of the pattern documented with the features that are common to all repetitions of the pattern in the original abstract plan. The resulting sequence of abstract operations and pattern clusters I call a *recovery plan skeleton*. (Take a look at figure 11, below, to see how it is).

4. Failure Recovery Planning

Failure recovery schemata can be thought of as conceptual categories, but not in the usual sense of the term. Although these categories organise similar concepts, their members cannot be enumerated. They can, however, be reconstructed as necessary. In the supervision system, when a failure is detected and the diagnosis function is able to produce the failure description, recovery planning is attempted. The first step is to look for similar failure situations previously encountered. Using the failed operation template plus the failure description template as indexing key, a failure recovery schema will eventually be retrieved. Then, the plan skeleton in that schema is adapted to solve the current situation.

4.1 Planning Guided by an Abstract Plan

Adapting a plan skeleton, so that it solves a particular situation, can be done by planning a solution for the situation and using the skeleton simply to guide the search. If the plan skeleton contains no clusters, an A* progression planning algorithm can be directly applied. In this case, all that must be done is to redefine the cost function to take into account the information contained in the plan skeleton. The previously extracted features play an important role in cost computations.

The following cost estimation scheme was implemented:

- At the root node of the search space, representing the failure situation, the entire plan skeleton is considered as the guide to reach a situation in which the failure is considered recovered. In each node of the search space there is some remaining part of the initial plan skeleton that provides guidance for going from that node to a solution node.
- When expanding a node, the first abstract operation in the part of the plan skeleton, which still remains to be traversed between that node and a solution node, is used to judge the utility of different operator applications and assign them different costs. Let OP be a legal operator application in the current node and let C_R be its real cost. Let AO be the first abstract operation in the current plan skeleton. The cost of OP actually considered by the planner, C_P, is defined by the following rules:
  - If OP is an auxiliary operation, therefore belonging to the *nil* class in the operator abstraction hierarchy (fig. 4), then C_P will be the real cost C_R.
  - If OP belongs to the category of AO, then let K be the number of features documenting AO in the plan skeleton and let V be the number of these features actually verified in the case of OP.

    In this case, we have $C_P = ((K+1)/(V+1)).\ C_R$. In this way, operator applications that verify a greater number of features of the abstract operation will be prefered.
  - In all other cases, C_P will be infinity.
- When a node n is created, its total cost is defined as $f(n) = g(n) + h(n)$, where: $g(n)$ is the sum of the C_P costs of all operations in the path leading from the root node to n; the cost of reaching a solution node, $h(n)$, is given by $h(n) = \alpha \cdot L$, where L is the length of the current abstract plan and α is the average number of operations in a task-level plan per each operation in the original abstract plan (typically $\alpha = 2.5$).

4.2 Expanding Plan Clusters

The planning process becomes more complex when the plan skeleton contains clusters, due to the uncertainty with respect to the number of cluster instances that are required for the particular problem. The approach proposed in [10], although considerably improving search complexity, when compared to unguided failure recovery planning, still suffers from a search complexity problem. In fact, an instance of a cluster with three abstract operations may be planned as a sequence of four, five or more task-level operations. Searching for the appropriate cluster instances may mean handling a search space with a depth of ten or twenty operations. Despite the features documenting the cluster, this may become unmanageable.

A better alternative is to determine, in the first place, the cluster instances that are required. However, since this problem interacts with the structure of the entire plan skeleton, the best approach is to carry out an abstract planning phase, in which the clusters in the plan skeleton are replaced by sequences of abstract operations and the parameters of the abstract operations are instanciated.

A greedy progression planning approach is followed in converting a plan skeleton with clusters into a mere sequence of abstract operator applications. When the planner encounters a cluster, it can either skip it or append a cluster instance to the current plan skeleton.

In this approach, cost is not taken into account. Only a measure of preference of a particular abstract operator application over alternative applications is used. Let *AO* be an abstract operation in the plan skeleton and let *K* be the number of features that documents it. Let *AA* be a possible application of *AO* and let *V* be the number of features in *AO* verified in *AA*. *AA* will be preferred over other operator applications if it has the highest value of the ratio $(V+1)/(K+1)$.

Note that, as a direct consequence of taking into account features in cost computations, recovery planning will be easier for those failure episodes that are more similar to the failure episode that originally enabled learning. Precisely the same occurs with human learning.

5. A More Complex Example

Fig. 8 illustrates a failure also due to a defect in a part, as in fig. 1. However, in this case, despite the defect, it was possible to assemble de part (side plate `sp1`) and two pegs (`peg1` and `peg2`). Execution only fails when the assembly of crossbar `cb1` is attempted. The failure description is given in fig. 9.

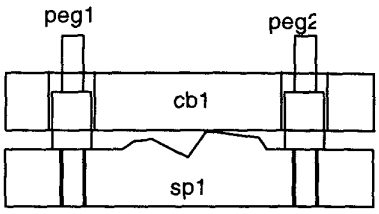

Fig. 8 — Defective assembly causing mate failure

```
failure_category(
 defective_assembly(obj66,sp1,cranf,fx),
 assemble(sony,gp_2,obj76,cb1,cb,cranf,fx),
   [ ], % delete list
   [ defective(obj66) ] % add list
).
```

Fig. 9 — Description of failure in fig. 8

The recovery strategy for this failure situation is given in fig. 10. In short, this 24 steps plan consists of disassembling the two spacer pegs (`peg1` and `peg2`), substituting the defective side plate by a new part of the same type, re-assembling the pegs and, finally, assembling the cross bar.

It is easy to realise that a planner, based on search only, could not find this 24 steps plan, in reasonable time. However, if an external agent provides the solution for the problem and its application is successful, this constitutes a learning opportunity. Applying the learning methodology presented in section 3, the failure recovery schema presented in fig. 11 is generated. (Features are given in italic).

```
Step #:
 1 putdown(sony,gp2,obj76)
 2 store_tool(sony,gp_2,tp_2)
 3 get_tool(sony,gp_3,tp_3)
 4 disassemble(sony,gp3,p1,peg2,peg,cranf,fix)
 5 putdown(sony,gp3,peg_o1)
 6 disassemble(sony,gp3,p2,peg1,peg,cranf,fix)
 7 putdown(sony,gp3,peg_o2)
 8 store_tool(sony,gp3,tp_3)
 9 get_tool(sony,gp1,tp_1)
10 disassemble(sony,gp1,obj66,sp1,sp,cranf,fix)
11 putdown(sony_scara,gp1,obj66)
12 feed_part(fd_sp,obj99,side_plate)
13 pick_from_feeder(sony,gp1,obj99,sp,fd_sp)
14 assemble (sony,gp1,obj99,sp1,sp,cranf,fix)
15 store_tool(sony_scara,gp1,toolp_1)
16 get_tool(sony_scara,gp3,toolp_3)
17 pickup_part(sony_scara,gp3,p1,peg)
18 assemble (sony,gp3,p1,peg2,peg,cranf,fix)
19 pickup_part(sony_scara,gp_3,p2,peg)
20 assemble (sony,gp3,p2,peg1,peg,cranf,fix)
21 store_tool(sony_scara,gp3,toolp_3)
22 get_tool(sony_scara,gp2,toolp_2)
23 pickup_part(sony,gp2,obj76,cb)
24 assemble(sony,gp2,obj76,cb,crossb,cranf,fix)
```

Fig. 10 - The recovery plan for the situation in fig. 8

```
[ assemble(R,T,Obj1,Comp1,Type1,Prod,Fix),
  defective_assembly(DefObj,Comp4,Type4,Prod,Fx),
  [ [ place(Obj1), [ ] ]
    [ [disassemble(X,C), place(X)],
        [ initially_assembled(C),
          assemble_after(C,Comp4),
          assemble_before(C,Comp1) ] ]
    [disassemble(DefectiveObj, Comp4), [ ] ]
    [place(DefectiveObj), [ ] ]
    [pick(NewObj),
        [ same_type_as(NewObj,DefectiveObj) ] ]
    [assemble(NewObj,Comp4),
        [ same_type_as(NewObj, DefectiveObj) ] ]
    [ [pick(X), assemble(X,C) ],
        [ initially_assembled(C),
          assemble_after(C,Comp4),
          assemble_before(C,Comp1)]]
    [pick(Obj1), [ ] ]
    [assemble(Obj1,Comp1), [ ] ]
]  ]
```

Fig. 11 — The failure recovery schema learned from the episode illustrated in figs. 8, 9 and 10

This recovery schema contains two clusters, one for the disassembly phase, `[disassemble(X,C), place(X)]`, and the other for the assembly phase `[pick(X), assemble(X,C) ]`, where variables `X` and `C` are constrained by the features. For instance, one of the features says that `C` represents an assembly component that must be assembled before `Comp1`, i.e. before the component whose assembly failed, leading to the need for a recovery strategy.

Finally, consider a situation in which this learned failure recovery schema can be applied. Suppose that, after assembling the side plate (`sp1`), the spacer pegs (`peg1`, `peg2`), the cross bar (`cb1`) and the shaft (`sft`), the robot fails to assemble the lever `lv` and realises that the failure was due, again, to a defect in the initial sideplate (`sp1`). The situation is similar to the previous one, but more complex, because there is a larger set of components that must disassembled in order to replace the defective

component. However, applying the planning strategy outlined in section 4, guided by the plan skeleton contained in the learned recovery schema, the solution for the new problem is easily derived (fig. 12). As can be seen, at the surface, the plan for the new problem is quite different from the plan used in the episode that enabled learning.

```
Step #:
 1 putdown(sony,gp_2,lv_o1)
 2 disassemble(sony,gp2,obj16,cb,crossb,cranf,fx)
 3 putdown(sony,gp_2,obj16)
 4 disassemble(sony,gp2,sft_o1,sft,shaft,cranf,fx)
 5 putdown(sony,gp_2,sft_o1)
 6 store_tool(sony,gp_2,tp_2)
 7 get_tool(sony,gp_3,tp_3)
 8 disassemble(sony,gp3,peg_o2,peg2,peg,cranf,fx)
 9 putdown(sony,gp3,peg_o2)
10 disassemble(sony,gp3,peg_o1,peg1,peg,cranf,fx)
11 putdown(sony,gp3,peg_o1)
12 store_tool(sony,gp3,tp_3)
13 get_tool(sony,gp1,tp_1)
14 disassemble(sony,gp1,obj66,sp1,sp,cranf,fx)
15 putdown(sony,gp1,obj66)
16 feed_part(fd_sp,obj99,side_plate)
17 pick_from_feeder(sony,gp1,obj99,sp,fd_sp)
18 assemble (sony,gp1,obj99,sp1,sp,cranf,fx)
19 store_tool(sony,gp1,toolp_1)
20 get_tool(sony,gp_3,toolp_3)
21 pickup_part(sony,gp3,peg_o2,peg)
22 assemble(sony,gp3,peg_o2,peg2,peg,cranf,fx)
23 pickup_part(sony,gp3,peg_o1,peg)
24 assemble(sony,gp3,peg_o1,peg1,peg,cranf,fx)
25 store_tool(sony,gp3,toolp_3)
26 get_tool(sony,gp2,toolp_2)
27 pickup_part(sony,gp2,sft_o1,shaft)
28 assemble(sony,gp2,sft_o1,sft,shaft,cranf,fx)
29 pickup_part(sony,gp2,obj16,crossb)
30 assemble(sony,gp2,obj16,cb,crossb,cranf,fx)
31 pickup_part(sony,gp2,lv_o1,lever)
32 assemble(sony,gp2,lv_o1,lv,lever,cranf,fx)
```

Fig. 12 - Plan built by following a learned schema

6. Conclusions

In this paper a method was proposed in order to solve failure recovery problems based on analogy with previous similar problems, whose solution was learned. The inefficiency of the classical planning-from-scratch approaches led robotics researchers to avoid artificial intelligence methods. However, as the method presented in this paper illustrates, the introduction of learning at the task planning and failure recovery level enables the system to make decisions in the time scale of the normal execution of the task. This means, that, if some unexpected problem arises and there is previous experience about problems of the same kind, the problem can be solved with the guarantee that the total time to complete the task won't increase significantly. Globally speaking, the learning by analogy approach enables the robot system to become increasingly effective, especially if learning occurs, not only within a task, but also across different tasks.

The proposed method uses deductive generalisation, as is common in EBL. However, in order to ensure operationality and utility of the learned representations, additional deductive (abstraction, feature extraction) and inductive (plan pattern clustering) processing is carried out. From CBR, the present work inherits the emphasis on the memorisation of cases. However, the cases stored in this system are not the descriptions of the failure recovery episodes themselves, but simply a summary of key aspects of the applied recovery solutions.

References

1. Camarinha-Matos, L.M., L. Seabra Lopes, J. Barata (1996) Integration and Learning in Supervision of Flexible Assembly Systems, *IEEE Transactions on Robotics and Automation*, vol. 12, p. 202-219.
2. Haigh, K.Z. and M. Veloso (1996) Interleaving Planning and Robot Execution for Asynchronous User Requests, *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS'96)*, Osaka, Japan, p. 148-155.
3. Kedar-Cabelli, S.T. and L.T. McCarty (1987) Explanation-Based Generalization as Resolution Theorem Proving, *Proc. 4th Int'l Conf. on Machine Learning*, Irvine, CA, p. 383-389.
4. Kolodner, J. (1993) *Case-Based Reasoning*, Morgan Kaufmann Publishers.
5. Michalski, R.S. (1994) Inferential Theory of Learning: Developing Foundations for Multitrategy Learning, *Machine Learning: a Multistrategy Approach IV*, R.S. Michalski and G. Tecuci (edrs.), Morgan Kaufmann Publishers.
6. Minton, S. (1990) Quantitative Results Concerning the Utility of Explanation-Based Learning, *Artificial Intelligence*, vol. 42, p. 363-392.
7. Mitchell, T., R.M. Keller, S.T. Kedar-Cabelli (1986) Explanation-Based Generalization: a Unifying View, *Machine Learning*, vol.1, p. 47-80.
8. Schraft, R.D. (1994) Mechatronics and Robotics for Service Application, *IEEE Robotics and Automation Magazine*, vol. 1, no. 4, p. 40-44.
9. Seabra Lopes, L. (1997) *Robot Learning at the Task Level: A Study in the Assembly Domain*, Ph.D. thesis, Universidade Nova de Lisboa, Portugal.
10. Seabra Lopes, L. and L.M. Camarinha-Matos (1996) Learning Failure Recovery Knowledge for Mechanical Assembly, *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS'96)*, Osaka, Japan, 2, p. 712-719.
11. Seabra Lopes, L. and L.M. Camarinha-Matos (1998) Feature Transformation Strategies for a Robot Learning Problem, *Feature Extraction, Construction and Selection: a Data Mining Perspective*, Kluwer Academic Publishers, p. 375-391.
12. Xue, G., T. Fukuda and H. Asama (1995) Error Recovery in the Assembly of a Self-Organizing Manipulator by using Active Visiual and Force Sensing, *Autonomous Robots*, 1, p. 179-186.
13. Yi, C and G.A. Bekey (1996) Assembly Planning for Modular Fixtures, *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, Osaka, Japan, p. 704-711.