Computers Math. Applic. Vol. 20, No. 9/10, pp. 101–110, 1990
Printed in Great Britain.

0097-4943/90 $3.00 + 0.00

LEARNING IN ORDER TO AVOID SEARCH IN LOGIC PROGRAMMING

M. J. PAZZANI
ICS Department, University of California, Irvine, CA 92717, U.S.A.

Abstract—This paper discusses learning in the context of a diagnostic expert system. The diagnostic expert system is an example of a *generate-and-test* problem solver. Fault diagnosis heuristics (i.e. logical implications representing association between unusual features and failed components) hypothesize potential faults. The potential faults are verified or denied by comparing the predictions of a qualitative simulation to observe data. Learning in this context consists of modifying the fault diagnosis heuristics. This paper describes how heuristic rules and device models can be represented and revised in a logic programming framework. In addition, we demonstrate how logic programming can be extended to perform abductive reasoning in addition to deductive reasoning. Finally, we compare failure-driven learning and learning from successes for acquiring fault diagnosis heuristics via explanation-based learning.

1. INTRODUCTION

Controlling search is a unifying theme of much work in artificial intelligence. There have been a number of techniques proposed in logic programming. The search strategy in Prolog, depth-first search with chronological backtracking is less than ideal for some algorithms. There have been a number of approaches suggested to augment this search strategy in a logic programming framework.

- Intelligent backtracking: by analyzing failures, search can continue at a choice point other than the most recent one [1–4].
- EPILOG: by co-routining the generation and testing of possibilities [5, 6].
- Re-ordering conjuncts: by testing those predicates that are more likely to fail first, search can be reduced since failure of a particular branch will be detected as soon as possible [7].

In this paper, I present an approach to learning to avoid unnecessary search. The approach is similar in spirit to intelligent backtracking and dependency-directed backtracking [8] with one important difference: rather than modifying the proof tree to get around the failure in the current case, the program is modified to detect the failure as early as possible in future similar cases.

The approach to learning has been implemented in the attitude control expert system (ACES) [9], a diagnostic expert system. ACES is designed to diagnose anomalies in the attitude control system of the DSCS-III satellite. ACES uses heuristic rules represented as logical implications to suggest possible faults. A fault is confirmed or denied by comparing the observed behavior to the behavior predicted by simulating the fault. Learning takes place in ACES when a heuristic rule proposes a fault which is denied by simulation. The heuristic that proposed the fault is revised by including the tests which the simulator used to rule out the fault. In similar future situations, the heuristic rule will not propose the fault. ACES uses explanation-based learning techniques [10, 11] to identify the conditions under which the heuristic will propose a fault which is denied. ACES is an example of a *generate-and-test* problem solver. Learning in this framework consists of moving predicates from the testing to the generation of hypotheses. This allows the fault to be ruled out as early as possible.

ACES was implemented in a version of Prolog that was written in LISP on a Symbolics 3600. We selected a logic programming approach in this project for a number of reasons. First, Prolog rules are ideally suited for implementing diagnosis heuristics.

Second, Prolog provides the flexibility to create meta-interpreters which can have different control strategies and can provide a trace of their reasoning. Explanation-based learning algorithms [12, 13] make use of a trace of rules used during a proof to determine how to generalize. Finally, the Prolog that we used allowed us to escape to Lisp to enable us to hook into existing I/O facilities.

There are several intrinsic difficulties in diagnosis problems. First, diagnosis can be viewed as a form of abductive inference [14] in which we are given an effect and must find a cause. Straightforward deduction would reason from causes to effects. In Section 2, we describe how logic programming can simulate a form of abductive reasoning.

Second, there are two basic approaches to diagnosis. Neither is entirely satisfactory. In one approach [15–17], the observed behavior of a device is compared to its predicted behavior which is specified by a quantitative or qualitative model of the device [18–20]. For a large system, such as a satellite, with a number of rapidly changing data values, comparing observed to predicted functionality can be inefficient. The alternative approach [21–23] encodes empirical associations between unusual behavior and faulty components as heuristic rules. This approach requires extensive debugging of the knowledge base to identify the precise conditions which indicate the presence of a particular fault.

In ACES, these two approaches are combined. Heuristic rules examine the atypical features and hypothesize potential faults. Device models confirm or deny hypothesized faults. Thus, heuristics focus diagnosis by determining which device in a large system might be at fault. Device models determine if that device is indeed responsible for the atypical features. When a fault is proposed, and later denied by device models, the heuristic which suggested the fault is revised so that the hypothesis will not be proposed in future similar cases. ACES learns after just one example how to avoid a hypothesis failure. It does this by finding the most general reason for the hypothesis failure. Device models explain how a hypothesis fails by finding those features that would be needed to confirm the hypothesis. Explanation-based learning improves the performance of ACES by creating fault diagnosis heuristics from information implicit in the device models. The basic idea behind the learning of fault diagnosis heuristics is that simulating a fault with device models will result in a number of predictions. If these predictions are not present in a particular system, the fault which was simulated can be ruled out. When a hypothesized fault is not confirmed, the heuristic that suggested the fault can be revised to not propose the fault in future similar cases. Learning heuristics by operationalizing information implicit in device models combines the best features of both these approaches. Heuristic rule application is more efficient than qualitative simulation. However, extensive debugging of the heuristic rules is not necessary because they are derived from a description of the device.

In this paper, we analyze the impact that two approaches to learning have on reducing the amount of search needed in fault diagnosis. In 1986, we implemented a failure-driven learning routine [24] which uses explanation-based learning to revise fault diagnosis heuristics that propose faults denied by qualitative simulation. In this paper, we compare this failure-driven learning strategy to an alternative strategy which learns from a successful problem solution by revising heuristic rules to include additional atypical features uncovered by a qualitative simulation. The point of this analysis is to demonstrate the benefits that can be derived from combining these two approaches to diagnosis.

Both approaches to revising heuristics make use of explanation-based learning for generalization. The difference between the approaches is *when* learning occurs and *what* is learned. Failure-driven learning adds additional conditions to rules which have proposed an incorrect hypothesis so that the same mistake will be avoided in the future. Success-driven learning adds conditions that indicate the situations in which the rule has proposed a confirmed fault.

Mostow and Bhatanger [25] have pointed out one advantage that explanation-based learning of failures has over explanation-based learning of successes: since the program learns as it searches it may be possible to solve problems which are too complex to solve otherwise. In this paper, we point out another advantage of explanation-based learning of failures: failure-driven learning acquires rules which indicate sufficient conditions for *not* exploring part of the search space. As a result, failure-driven learning results in a reduction in the search space of future problems (as measured by the number of hypotheses generated). In the remainder of this paper, we first present a logical analysis of the diagnosis task and illustrate how it can be cast as a generate-and-test

problem in logic programming. Next we present an overview of the two approaches to learning and report on the results of an experiment that compares the learning strategies. Finally, we indicate some directions for future research.

2. DIAGNOSIS IN ACES

ACES was designed to process telemetry data from a satellite and isolate the cause of problems with the attitude control system. The telemetry stream is processed by extracting symbolic features from numeric data representing "atypical" value for signals as well as atypical changes in values. A short explanation of the attitude control system and some potential faults will help to understand the results.

The attitude control system is responsible for detecting and correcting deviations from the desired attitude (i.e. orientation) of the satellite. To correct any deviations from the desired attitude, the attitude control system issues drive signals to the motors of four reaction wheels to change the wheel speeds. (Changing the wheel speed changes the stored momentum of the satellite and results in a change in the attitude of the satellite.) The wheel speeds are measured by a set of tachometers. The change in the wheel speed is combined with the information from sun and earth sensors to estimate the attitude of the satellite. When the estimated attitude deviates from the desired attitude, the attitude control system adjusts the wheel speeds to reorient the satellite.

ACES contains an initial set of fault diagnosis heuristics that suggest explanations for atypical features in the telemetry stream. For example, one heuristic indicates that if a tachometer of a reaction wheel is reading 0, then the tachometer might be broken. This rule is illustrated in Fig. 1. This rule is in a version of Prolog implemented in Lisp. The first element of a list is the predicate name and all variables are preceded by "?". The part of the rule preceded by "<-" is a fault hypothesis, and the part of the rule after "<-" is the conditions that are necessary to be proved to propose the hypothesis.

The initial fault diagnosis heuristics are definitions of faults. For example, a broken tachometer is one that is reading 0. These heuristics suggest faults to focus the model-driven diagnosis. Model-driven diagnosis consists of a qualitative simulation that matches the implications of a fault against observed data.

Figure 2 presents a second diagnostic heuristic and raises an interesting issue. This heuristic has the conditions as the heuristics in Fig. 1, but a different conclusion.

These rules illustrate an intrinsic problem in diagnosis. We are attempting to infer an unobservable explanation for observed symptoms. There may be multiple explanations for a set of observed symptoms. It is not correct to represent these rules as logical implications. A fault may imply a set of features, but a set of features does not necessarily imply a fault. A correct, but less useful implication is given in Fig. 3.

```
(problem (broken-wheel-tach ?wheel ?from)) <-
         (feature(value-violation ?sig ?from ?until 0))
         (measurement  ?sig ?wheel speed ?tach)
         (isa ?wheel reaction-wheel)
```

Fig. 1. A fault diagnosis heuristic: a tachometer of a particular wheel is broken if the wheel speed of that wheel reads 0.

```
(problem (broken-wheel-drive ?wheel ?from)) <-
         (feature(value-violation ?sig ?from ?until 0))
         (measurement  ?sig ?wheel speed ?tach)
         (isa ?wheel reaction-wheel)
```

Fig. 2. A second fault diagnosis heuristic: a wheel drive is broken if the wheel speed of that wheel reads 0.

```
(feature(value-violation ?sig ?from ? 0)) :-
         (problem (broken-wheel-drive ?wheel ?from))
         (measurement  ?sig ?wheel speed ?tach)
         (isa ?wheel reaction-wheel)
```

Fig. 3. Reasoning from failures to symptoms: a wheel speed reads 0 if the wheel drive reads 0.

The rule in Fig. 3 is logically correct. If the motor that causes a wheel to spin is broken, the speed will read 0. It reflects the fact that each failure implies (or causes) a unique set of symptoms. However, the rule in Fig. 3 is not useful for diagnosis since it deduces symptoms given a fault. The real problem is to infer a fault given a set of symptoms. This cannot be accomplished by deductive reasoning, but requires abduction [26, 27].

Deduction follows the following rule of inference, called *modus ponens*:

Given: IF E then H

H

In logic programming, this logical implication IF E then H is written as:

H:-E

Abduction follows a different rule of inference:

Given: IF H then E
E

H (maybe)

In Prolog, we represent this as a IF H then E a heuristic rule and implement a meta-interpreter to handle abduction:

H<-E

The meta-interpreter is similar to *modus ponens* with the addition that it checks the consistency of the hypothesis:

Given: H<-E
E
consistent(H,M,O)

H

Here, the term consistent(H,M,O) means that the hypothesis H must be consistent with the observed data O and the model of the device M as modified by the hypothesis. In ACES, the rule H<-E corresponds to a fault diagnosis heuristics that generates a hypothesis, the term consistent(H,M,O) corresponds to testing the hypothesis against a model of the device. This is exactly the hypothetico-deductive model of reasoning [28]. First, a hypothesis is generated, and then it is evaluated by deducing its implications and evaluating these deductions against observations. Figure 4 illustrates a rule, represented as a logical implication that predicts one result of a broken wheel drive.

The idea behind combining heuristic rules and device models is to facilitate the entry of knowledge into the system. It is relatively easy to find one symptom for any failure. This symptom serves as the initial diagnosis heuristic. Similarly, a model of a functionality and connectivity of a system is often produced at the time the system is designed, and it is relatively straightforward to convert this model to a machine understandable form. Learning diagnosis heuristics from device models avoids the need for an expert to identify the exact conditions that distinguish two faults. Instead, these are learned in the course of solving problems. In Ref. [9], we reported that ACES was able to learn heuristics that are slightly more efficient than those obtained by interviewing a domain expert.

```
(predict (broken-wheel-drive ?wheel ?time)
         (jump ?sig ?time ? ? 0 ?slope)) :-
    (friction-constant ?wheel ?slope)
```

Fig. 4. A prediction made about the effects of a broken wheel drive. If the wheel drive motor is broken on ?wheel at ?time, then there will be a jump in the wheel speed at 0. The slope of the jump at 0 will be equal to the friction constant of the wheel (i.e. friction will stop the wheel if the motor breaks).

3. THE EFFECTS OF LEARNING

The goal of learning in ACES is to make the diagnosis process more efficient. Initially, ACES may require a great deal of search to arrive at a correct diagnosis. The major source of this problem is that the initial definitional diagnosis heuristics are overly general. Often, there are several heuristics which hypothesize different faults in a given situation. For example, there are two rules which suggest an explanation for a tachometer reading of 0. The first rule, illustrated in Fig. 1 blames the tachometer. Another rule, illustrated in Fig. 2, blames the motor causing the wheel to spin. A simulation can distinguish these two faults because they have different effects on the behavior of other satellite components. For example, a broken wheel drive motor will result in a slow decrease in the speed of the wheel due to friction until the wheel stops. A broken tachometer will result in a change of attitude in the axes controlled by the wheel.

3.1. Failure-driven learning

When a diagnosis heuristic proposes a fault hypothesis that is denied because it is not consistent with derived implications, failure-driven learning can revise the heuristic so it does not propose the same fault in future similar conditions. By "similar", we mean those conditions that would result in the same inconsistency. Recall that an inconsistency is detected when a prediction derived from the device is not substantiated by the observed data. To avoid this inconsistency the diagnosis heuristic is revised to check for this prediction (under the set of initial conditions found to be relevant by explanation-based learning) before generating the fault. The effect of failure driven-learning is that a condition is acquired which distinguishes the hypothesized fault from whatever the fault is in this situation. This condition is a sufficient condition for ruling out the fault. Note that this does not imply that the satellite be in an identical state to rule out the fault.

3.2. Learning from successes

A rule may be modified after the hypothesis it has suggested has been confirmed by device models. This modification may make it more efficient to recognize instances of that fault in the future. Most explanation-based learning systems (e.g. Refs [12, 13]) operate from successes rather than failures. There are at least three possibilities.

First, the confirmation process consists of verifying that a number of implications of a proposed fault are present in the observed data. Each implication is a necessary condition for confirming the fault. Taken together, all of the conditions are sufficient for the system to report that the fault is consistent with the observed data. One approach to modifying a heuristic rule would be to create a chunk [29] or macro-operator [30, 31] that tests the same conditions as the generation and testing of a particular hypothesis. In this approach, a fault diagnosis heuristic is modified to include all of the tests performed in the confirmation process. This approach can result in performance improvements because explanation-based learning operationalizes [32, 33] the testing of these conditions. This technique is called *SuccessAll* in the section on empirical results.

Second, a more elegant approach to moving conditions from the testing process to the generation process is to move only those conditions that do not describe the normal operation of the system. These conditions will distinguish the particular fault from a properly operating system. This technique is called *SuccessDifferent* in the section on empirical results. A similar strategy has been proposed by Steels and Van de Velde [34]. However, Steels and Van de Velde's strategy makes use of empirical rather than explanation-based learning techniques.

Finally, a simpler approach is to select one prediction of the simulated fault at random to incorporate into the diagnosis heuristic. This technique is called *SuccessRandom* in the section on empirical results.

In PRODIGY [35], revisions acquired via explanation-based learning of successes are treated as preferences. In future processing, rules whose preferences are satisfied are tried before other rules. In ACES, the order of search is not important because the entire search space must be explored to return all solutions which are consistent with the data.

FAULT	NO	Fail	All	Diff	Rand
Tachometer	2268	211	2026	14870	16
Wheel Drive	1236	616	1089	998	1286
Wheel Drive (Equal)	745	469	640	837	721
Unload	870	861	766	861	865

Fig. 5. Number of hypotheses generated before and after learning.

FAULT	NO	Fail	All	Diff	Rand
Tachometer	21	1	1	67	16
Wheel Drive	4	1	1	2	4
Wheel Drive (Equal)	2	1	1	2	4
Unload	1	1	1	1	1

Fig. 6. Number of logical inferences before and after learning.

4. EMPIRICAL RESULTS

In this section, we compare failure-driven learning to the variations of success-driven learning. We ran each of the learning strategies on four test cases and then recorded how learning affected the performance of the system on the same four test cases.† The four test cases are:

- A tachometer stuck at 0. The implications of this failure are that the attitude of the satellite will be disturbed outside of normal operating ranges and all of the wheel speeds will change more rapidly than usual.
- A wheel drive motor ignoring its input when the opposite wheel is at the same speed.‡ Friction will cause the wheel speed to slow down and stop. The speed of the opposite wheel will go to 0 so that the difference between the wheel speeds remains constant. (Since the momentum remains constant, the attitude will not be disturbed by the failure of one wheel.)
- A wheel drive ignoring its input in the usual case where the opposite wheel is at a different speed. Friction will cause one wheel to stop and the opposite wheel changes speed but does not go to 0.
- A wheel unloaded (i.e. the speed of the reaction wheels is changed by the firing of a thruster). This is not actually a failure, but it changes the wheel speeds at momentum so rapidly that the monitor detects atypical features. These atypical features must be explained in the same manner that a fault is explained.

There are two metrics for comparing the effects of learning on ACES. The first metric is the number of hypotheses generated (see Fig. 5). The second metric is the number of logical inferences required to generate and confirm a solution (see Fig. 6).

The results here illustrate a number of points about the various learning strategies.

As illustrated by Fig. 5, both *FailureDriven* and *SuccessAll* revise the heuristic rules so that the confirmation process does not reject a hypothesis that has been generated (i.e. only one hypothesis is generated by the heuristic rules). The reason that this occurs is different for each case. The *SuccessAll* strategy creates a chunk which in effect does the generation and tests a hypothesis in one step. However, the number of logical inferences does not decrease as dramatically as the number of hypotheses. In effect, what occurs is that much of the search is moved from the testing to the generation of hypotheses. In contrast, the *FailureDriven* strategy reduces the number of hypotheses generated by moving only one test from the confirmation to the generation process. This test distinguishes one fault from another fault with similar features.

The number of logical inferences required to solve the problem decreases because there are parts of the search space which do not need to be explored. Figure 7 illustrate the search space of the problem before learning. For this problem ACES initially blames a wheel speed of 0 on a broken tachometer. However, this hypothesis is ruled out because a broken tachometer would result in

†Ideally, one would want to test the system on cases which differ from the training examples. Unfortunately, we have only been able to collect a small number of examples of telemetry data. We have also tested the system on hand-generated descriptions of telemetry data and received similar results to those reported here.

‡There are four reaction wheels in the attitude control system. Pitch and roll momentum are represented as the difference in wheel speed of two opposing wheels.

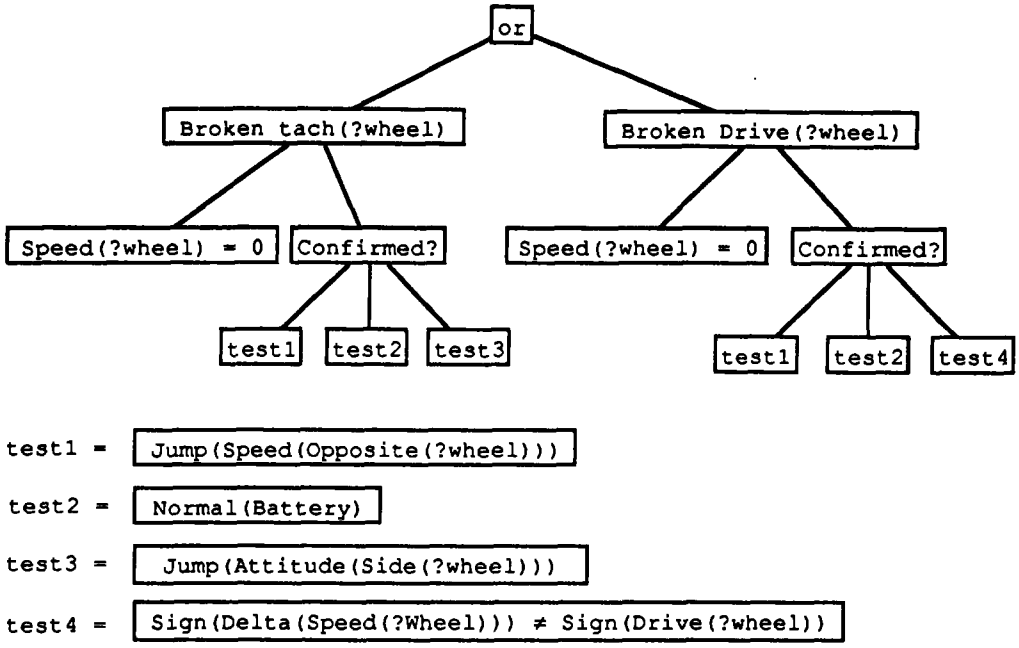

Fig. 7. An overview of the search space explored before learning to determine that a wheel drive motor is broken.

a feature (a change in the attitude) which was not observed. A second hypothesis, a broken wheel drive motor is proposed and confirmed. This is the third problem in Figs 5 and 6.

In the *SuccessAll* strategy, the "Confirmed?" step is replaced by the tests that "Confirmed?" would make. Some efficiency is gained because "Confirmed?" must decide what tests to perform and then performs the tests. After learning, the decision as to what tests are to be performed is also avoided since these are "compiled out" by the learning process. Figure 8 illustrates the search space after learning with the *SuccessAll* strategy.

Figure 8 illustrates the search space for the same problem after using the *FailureDriven* strategy. This strategy moves a condition from the confirmation process to the test process when the test generates a hypothesis which fails to be confirmed. The condition is one which would have to be true if the hypothesis were to be true. This condition distinguishes the hypothesized fault from a different fault with similar features. For example, in ACES, a wheel speed of 0 is the initial symptom of a broken tachometer and of a broken wheel drive. After failure-driven learning, an additional condition is added to the tachometer rule to check for "Test-3", a change in the attitude. This distinguishes a broken tachometer from a broken wheel drive. Therefore, as illustrated by Fig. 9, ACES no longer hypothesizes a broken tachometer when a wheel drive is broken.

Both the *SuccessRandom* and the *SuccessDifferent* strategy add conditions to the rules which generate hypotheses. However, these conditions are not guaranteed to rule out any other fault. For example, the *SuccessDifferent* strategy can change the tachometer heuristic to check for a jump in the opposite wheel speed. Unfortunately, this is also a sign of a broken wheel drive motor so this additional condition does not rule out a broken wheel drive. Figure 10 shows the changed rule.

As a consequence, after learning with the *SuccessDifferent* strategy, the heuristics still hypothesize

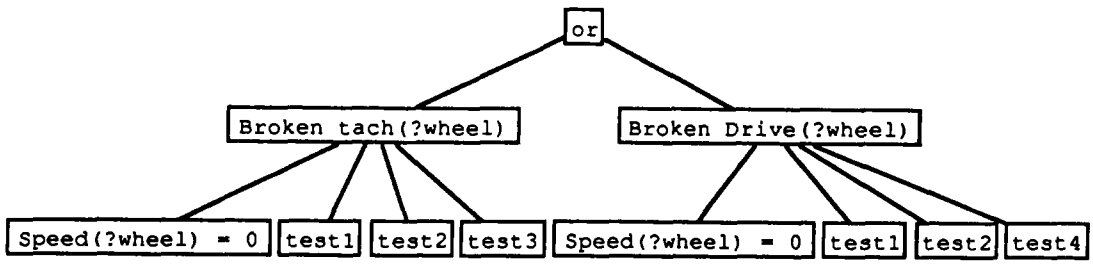

Fig. 8. An overview of the search space explored after learning with the *SuccessAll* strategy.

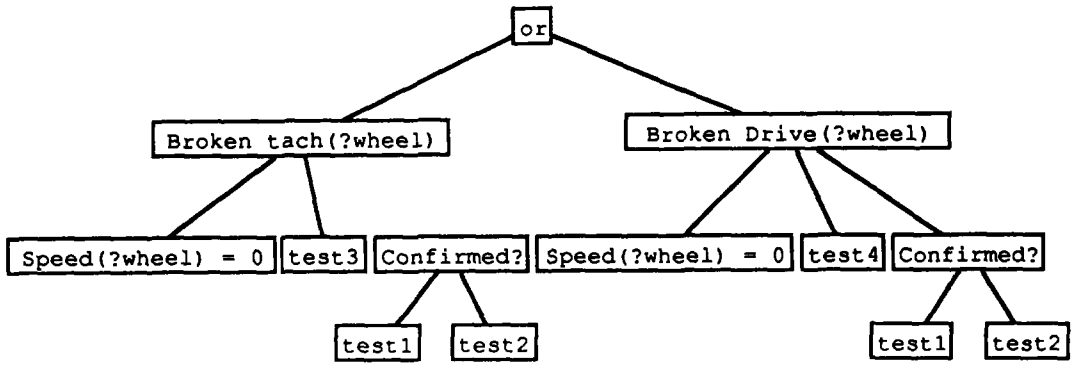

Fig 9. An overview of the search space explored after learning with the *FailureDriven* strategy. The "Confirmed?" branch is avoided because "Test-3" fails.

```
(problem (broken-wheel-tach ?wheel ?from))
      (feature(value-violation ?sig ?from ?until 0))
      (measurement  ?sig ?wheel speed ?tach)
      (isa ?wheel reaction-wheel)
      (opposite-wheel ?wheel ?wheel-098)
      (measurement  ?sig-099 ?wheel-098 speed ?device-100)
      (abstract-feature (jump ?sig-099 ? (after ?from)
                              ? ? ? ?))
```

Fig. 10. The tachometer heuristic revised by the *SuccessDifferent* strategy. After the wheel speed goes to 0, the speed of the opposite wheel must also change.

a broken tachometer in addition to a broken wheel drive (see Fig. 11). A similar problem can arise in the PRODIGY system which treats the additional conditions as preferences. This problem occurs because there is no guarantee that a condition (or preference) added to one rule will distinguish that rule from others. To address this issue, PRODIGY evaluates the utility of the search control information acquired through learning. If the cost of making a test is greater than the benefit then the test is not made.

One unusual data point in Figs 5 and 6 deserves further explanation. In the first test case, the number of hypotheses and number of inferences increases after learning in the *SuccessDifferent* strategy. This is caused by adding a condition to the wheel unload heuristic. The condition (an attitude disturbance) occurs several times after a broken tachometer. This causes ACES to hypothesize a wheel unload when a tachometer is broken for each way of satisfying the wheel unload heuristic.

4.1. Summary of empirical results

We draw the following two conclusions from our analysis:

- Failure-driven learning finds sufficient conditions for ruling out a fault.
- Success-driven learning finds sufficient conditions for establishing a fault. These conditions need not rule out other faults.

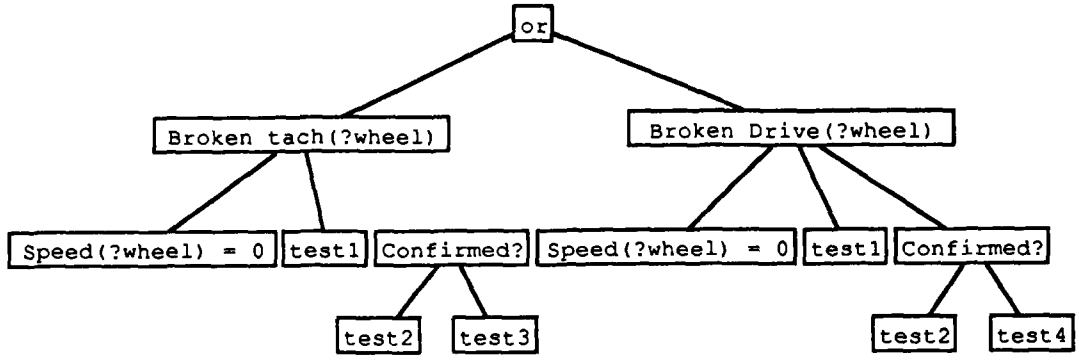

Fig. 11. An overview of the search space explored after learning with the *SuccessDifferent* strategy

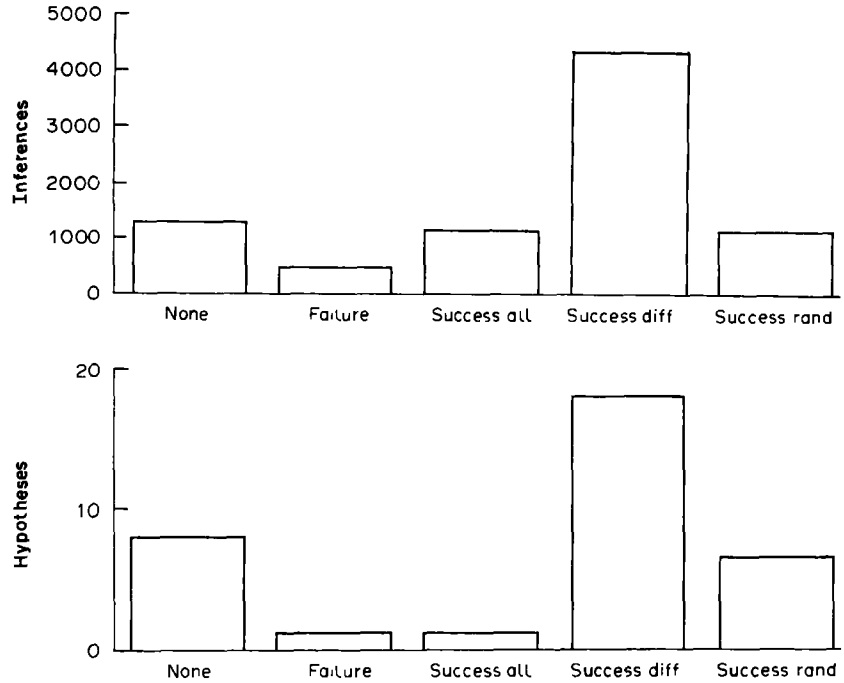

Fig. 12. Average number of hypotheses and logical inferences before and after learning for the four learning strategies.

This analysis is supported by the experiments we have run. The results of our experiments are summarized in Fig. 12.

5. LIMITATIONS AND FUTURE WORK

These experiments have demonstrated that for certain tasks such as fault diagnosis, failure-driven learning increases performance more than several variants of success-driven. The primary reason for this is that failure-driven learning modifies a heuristic by adding only those conditions that distinguish one fault from another.

These results are not limited to diagnosis tasks. The crucial feature of the diagnosis task is that all consistent solutions are found. If only one solution were needed (e.g. finding a plan to achieve a goal), then success-driven learning which finds preferences to order the search might create heuristics which locate one solution more efficiently than failure-driven learning. More research needs to be done in this area.

Another area for future research is the ordering of multiple conditions added to rules. Multiple revisions occur in failure-driven learning if a hypothesized fault fails for different reasons on different training examples. The condition which is more likely to fail should be tested first. Keller has proposed including a set of representative examples as an additional input to a learning system [36]. If such a set of examples were input to ACES, then it would be possible to determine which test fails more often. An alternative approach would be to dynamically monitor the performance system.

Finally, different approaches to avoiding search in a logic programming framework should be explored. For example, a success-driven approach may learn the concept "the set of all possible failures" given a set of conditions. Then, it would not be necessary to search all paths to find this set. In this case, preferences for the most commonly encountered sets of failures may result in less search than a failure-driven approach.

6. CONCLUSION

We have compared strategies of learning from failures to learning from successes in the context of a *generate-and-test* problem solver. One result of the paper is fairly straightforward: Failure-driven learning creates rules which distinguish between failures. This is demonstrated by the fact

that the number of hypotheses decreases after learning. A more subtle result is that the performance of the system, measured in terms of logical inferences decreased with failure-driven learning more than it did with two variants of success-driven learning. We have tested our approach on an example of fault diagnosis, but we believe that the approach will extend to any problem that can be specified as a generate-and-test problem in a logic programming framework.

REFERENCES

1. M. Bruynooghe and L. Pereira, Deduction revision by intelligent backtracking. *Implementations of Prolog* (Ed. J. A. Campbell), Ellis Horwood, Chichester (1984).
2. M. Bruynooghe, Solving combinatorial search problems by intelligent backtracking. *Inf. Process. Lett.* **12**(1), (1981).
3. P. Cox and T. Pietrzykowski, Deduction plans: a basis for intelligent backtracking. *IEEE Trans. Pattern Analysis Mach. Intell* **3**(1), (1981)
4. P. Cox, Finding backtracking points for intelligent backtracking *Implementations of Prolog* (Ed. J. A. Campbell), Ellis Horwood, Chichester (1984).
5. L. Pereira, Logic control with Logic. *Implementations of Prolog* (Ed. J. A. Campbell), Ellis Horwood, Chichester (1984).
6. A. Porto, EPILOG: A language for extended programming in logic. *Implementations of Prolog* (Ed. J. A. Campbell) Ellis Horwood, Chichester (1984)
7. L. Naish, Prolog control rules. *9th Joint Conf. Artificial Intelligence* (1985).
8. R. Stallman and G. Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artif. Intell.* **9**(2), 135–196 (1977).
9. M. Pazzani, Failure-driven learning of fault diagnosis heuristics. *IEEE Trans. Systems Man Cyber.* **17**(3), 380–394 (1987).
10. T. Mitchell, S. Kedar-Cabelli and R. Keller, Explanation-based learning: a unifying view. *Mach. Learning* **1**(1), (1986).
11. G DeJong and R. Mooney, Explanation-based learning: an alternate view. *Mach. Learning* **1**(2), (1986).
12. H. Hirsh, Explanation-based learning in a logic programming environment. *Proc Int. Joint Conf Artificial Intelligence, Milan, Italy*. Morgan Kaufmann, Los Altos, Calif. (1987).
13. S. Kedar-Cabelli, Explanation-based generalization as resolution theorem proving. *Proc. 4th Int. Machine Learning Workshop*, Irvine, Calif. (1987)
14. J. Josephson, B Chandrasekaran, J Smith and M. Tanner, A mechanism for forming composite explanatory hypotheses. *IEEE Trans. Systems Man Cyber.* **17**(3), 445–454 (1987)
15. R. Davis, H. Shrobe, *et al.*, Diagnosis based on description of structure and function. *Natn. Conf. Artificial Intelligence* American Association for Artificial Intelligence, Pittsburgh, Pa (1982).
16. M Genesereth, J. S. Bennett and C. R. Hollander, DART: expert systems for automated computer fault diagnosis. *Proc. Ann. Conf. Association for Computing Machinery*, Baltimore, Md (1981).
17. E A. Scarl, J. Jamieson and C. A. Delaune, A fault detection and isolation method applied to liquid oxygen loading for the space shuttle. *Proc. 9th Int. Joint Conf. Artificial Intelligence*, Los Angeles, Calif. (1985).
18. B. Kuipers, Commonsense reasoning about causality: deriving behavior from structure. *Artif. Intell.* **24**, 1 (1984).
19 K. Forbus, Qualitative process theory. *Artif. Intell.* **24**, 1 (1984).
20. J. de Kleer and J. Brown, A qualitative physics based on confluences. *Artif. Intell.* **24**, 1 (1984).
21. E. H. Shortliffe, *Computer-based Medical Consultation: MYCIN* Elsevier, New York (1976).
22 W. R. Nelson, REACTOR: an expert system for diagnosis and treatment of nuclear reactor accidents. *Proc. Natn Conf. Artificial Intelligence*, AAAI, Pittsburgh, Pa (1982).
23. R. E. Wagner, Expert system for spacecraft command and control. Computers in aerospace *IV Conf. Am. Inst Aeronautics and Astronautics*, Hartford, Conn. (1983).
24 M. Pazzani, Refining the knowledge base of a diagnostic expert system: an application of failure driven learning. *Proc. Natn. Conf. Artificial Intelligence*. American Association for Artificial Intelligence (1986).
25 J. Mostow and N. Bhatnager, Failsafe—a floor planner that uses EBG to learn from its failures. *Proc 10th Int. Joint Conf. Artificial Intelligence*, Milan, Italy (1987).
26. E. Charniak and McDermott, *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, Mass. (1985).
27. P. O'Rorke, Abduction and Machine Learning. *AAAI Symp. Explanation-based Learning*, Stanford, Calif. (1988).
28. R. Patil, Artificial intelligence techniques for diagnostic reasoning in medicine *Exploring Artificial Intelligence* (Ed. H Shrobe). Morgan Kaufmann, Los Altos, Calif. (1988).
29 J. Laird, P Rosenbloom and A. Newell, Chunking in soar. the anatomy of a general learning mechanism. *Mach. Learning* **1**, 11–46 (1986).
30 R. E. Korf, Macro-operators: a weak method for learning. *Artif Intell.* **26**, 1 (1985)
31. R Fikes, R. Hart and N Nilsson, Learning and executing generalized robot plans. *Artif. Intell.* **3**, 251–288 (1972).
32 R. Keller, Defining operationality for explanation-based learning. *Proc. Natn. Conf. Artificial Intelligence*, Seattle, Wash., pp 482–487 (1987).
33 J Mostow, Searching for operational concept descriptions in BAR, MetaLex, and EBG. *Proc. 4th Int. Machine Learning Workshop*, Irvine, Calif, pp. 376–382 (1987)
34. L. Steels and W Van de Velde, Learning in second generation expert systems. *Knowledge-Based Problem Solving* (Ed. J. S. Kowalik), Prentice-Hall, Englewood Cliffs, N.J (1986)
35 S. Minton, J. Carbonell, O. Etzioni, C. Knoblock and D. Kuokka, Acquiring effective search control rules: explanation-based learning in PRODIGY. *Proc. 4th Int. Machine Learning Workshop*, Irvine, Calif. (1987).
36. R. Keller, Concept learning in context. *Proc. 4th Int. Machine Learning Workshop*, Irvine, Calif., pp 482–487 (1987).