Task Structures: What To Learn?

Eleni Stroulia and Ashok K. Goel *
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
eleni@cc.gatech.edu, goel@cc.gatech.edu

Abstract

Broadly characterized, learning can improve problem-solving performance by increasing its efficiency and effectiveness, and by improving the quality of produced solutions. Traditional AI systems have limited the role of learning to the first two performance-improvement goals. We have developed a reflection process that uses a model of the system's functional architecture to monitor its performance, suggest a quite broad range of modifications when it fails, and subsequently perform these modifications to improve its problem-solving mechanism. The modifications suggested and performed by the reflection process may result in performance improvement of all the above types.

Introduction

Simon [21] characterizes the goal of learning as improving the performance of an intelligent system on some reasoning task. In his characterization, learning may result in improvements in the system's problem-solving effectiveness (successfully solving more problems in a given population of problems), or in its problem-solving efficiency (more rapid response, use of fewer computational resources). In addition, learning may result in an improvement in the quality of solutions produced by the system (producing more optimal solutions).

If the task environment of the system is static, and the system's knowledge of the environment is complete and correct, then we (as AI designers) may a-priori assume fixed measures of problem-solving effectiveness, computational efficiency, and solution quality. These assumptions, however, do not hold in realistic task environments. In general, the task environment can be dynamic, giving rise to new requirements, new constraints and new opportunities over time. The new requirements may result in new problems that the system has to address, and the new constraints and opportunities may result in new measures of problem-solving effectiveness, efficiency, and solution quality. Further, if the task environment is dynamic, then the assumption about the completeness and correctness of the system's knowledge cannot hold for long. This means that the system would need to acquire and assimilate new knowledge of its environment.

Given the above relationship between learning goals and problem-solving performance, one method for learning is to produce a solution to a problem (if possible), evaluate the solution (if possible), reflect upon the problem solving, assign credit (or blame) for the success (or failure) of problem solving, and, if needed, appropriately modify the problem solving to improve its performance. Indeed, performance-driven methods of learning have received much attention in AI research on learning [20, 22, 17, 2, 12]. However, much of this work accommodates only some kinds of modifications to problem-solving, and, thus, addresses only some of the above learning goals. Let us consider Samuel's [20] checkers-playing program and Mitchell et al's [17] symbolic-integration program as two illustrative examples. These two programs share several attributes. Firstly, both programs assume complete knowledge of their task environment in the form of allowed operations (e.g., legal operations in checkers, legal operations of simplification and integration). Secondly, both systems modify only their heuristics for selecting among the applicable operations at a given state, and, thus, improve their effectiveness (e.g., the number of games they can win, or the number of integration problems they can solve), and their efficiency (the number of moves in a game, or the number of symbolic-manipulation steps in a solution).

Note, however, that the learning process does not result in modifications to the functional architecture of these two systems. The functional architectures and reasoning mechanisms of the two systems are quite similar, consisting essentially of two elements: one decides which operations are legally applicable at a given state, and the other decides which one of the legal operations is the most appropriate to apply. Learning does not affect the first element since the rules of checkers and the set of symbolic-integration operators are non-modifiable. It only affects the second element in that it modifies the knowledge (evaluation functions, applicability concepts) it uses. However, no new elements are added to the functional architec-

*This work has been supported by a research grant from the Office of Naval Research, contract N00014-92-J-1234, a graduate fellowship from IBM, a research gift from NCR, and equipment donated by IBM and Symbolics.

ture, the interaction between the existing ones does not change, and neither does the way these elements use the existing knowledge. In summary, both systems' performance improves only in terms of effectiveness and efficiency; the mechanism that carries out their problem solving is simple, and it does not get greatly affected by the learning process.

In dynamic environments, the scope of learning is much more broader: it is to enable the system to sustain quality performance over time. Essentially, learning has to be able to modify the system's architecture to "pay attention" to the evolving domain, and its opportunities and requirements. To that extent, learning has to accommodate more interesting modifications to the system's functional architecture: first, new domain knowledge may have to be acquired regarding the new domain properties. Because the environment is dynamic, it is not realistic to assume that all possible types of domain knowledge can be thought off a priori; therefore it is possible that some new domain knowledge may not fit with the system's view of the domain. Such incongruence may necessitate modifications to the system's domain-representation scheme adopted. Second, the system's existing knowledge may have to be reorganized in relation to its newly acquired parts. Finally, and even more importantly, the elements of the system's functional architecture (and the interactions among them) may have to be redesigned to make use of the newly acquired knowledge.

These types of learning tasks, modifications to the system's problem-solving mechanism, require

1. an explicit model of the interactions among the representation of the system's knowledge, its content, its organization and the way it is used by its problem-solving mechanism,

2. a language for capturing the above model, and

3. a process which can use the model of the system's problem-solving mechanism, to monitor its performance, and, when it fails, suggest and perform modifications to it, which can improve its performance.

In our work, we have adopted as an initial modeling framework for capturing these interactions the task-structure analysis of problem solving [3]. Further, we have adapted the language of Structure-Behavior-Function (SBF) models of how physical devices work [8] to describe how problem solvers work. Furthermore, we have developed a reflective process (consisting of a monitoring task, a blame-assignment task, and a and repair task) which is able to suggest and perform a class of modifications to the system's problem-solving mechanism which result in improving the system's in terms of effectiveness, efficiency, and quality of solutions produced.

Task Structures: A language for describing Problem Solving

Recent work on problem solving has led to a family of theories [4, 28, 15, 3, 23] that explicitly represent the interactions between the knowledge and the problem-solving mechanism that uses it. We adopt task structures [3] as a framework for analyzing and modeling problem solving. In this framework, a task is specified by the type(s) of information it consumes as input, the type(s) of information it produces as output, and a generic task of which it is an instance. A task can be accomplished by one or more methods, each of which may decompose it into a set of simpler subtasks. A method is characterized by the kinds of knowledge it uses, the subtasks it sets up, and the control it exercises over the processing of these subtasks. These subtasks can, in turn, be accomplished by other methods, or, if the appropriate knowledge is available, they may be solved directly. The task structure of a problem solver thus provides a recursive decomposition of its overall task in terms of methods and subtasks.[1]

SBF models

To describe the task structure of a problem solver, we have adapted the language of structure-behavior-function (SBF) models for describing how physical devices work [8, 7, 26], to describe how the problem-solver's task structure works. In this language, a problem-solving task is a transformation from an input information state to an output information state. It is annotated by a set of conceptual relations between the input and output information, a pointer to the prototypical task of which it is an instance, and pointers to the methods than can potentially accomplish it. The task's conceptual relations constitute a partial description of the expected, correct performance of the task. If the task is accomplished by a method, the conceptual relations of the method's subtasks and the ordering relations it imposes over them constitute a partial description of a correct problem-solving mechanism for this task.

In an extension of the task-structure framework, the SBF model of a problem solver also includes a meta-level description of its domain knowledge, in terms of the types of known domain objects and the relations applicable to them. The types of information that the problem solver consumes and produces are related to the types of domain objects, and the conceptual relations of its tasks may refer to relations among these objects.

The Taxonomy of Learning Tasks

The task-structure view of problem solving gives rise to a taxonomy of learning tasks (modifications to the problem-solving mechanism) that a problem solver can set up for itself. In this section, we discuss each one of these modifications to the problem-solving mechanism with respect to the kinds of performance improvement they lead to. Often the accomplishment of a learning task gives rise to some new learning tasks, therefore we also point out the interactions between them.

[1] Tasks can be thought of as roughly equivalent to goals, and "leaf" tasks (that is, tasks not further decomposable by methods) are similar to elementary goals which can be immediately accomplished by operators. Methods can be thought of as general plans for how the solutions to different subgoals combine to accomplish higher-level goals.

Modifying a task

A task is defined in terms of the types of information it takes as input, the types of information it produces as output, and a set of conceptual relations between them. Therefore, to modify a task one can modify either the types of information it consumes, or the types of information it produces, or the transformation it performs. The modification of either of these aspects results in modification of the class of problems solvable by this task. The modification of the task's input information-type modifies the domain to which a task is applied. The modification of the task's output information-type modifies the range of outputs the task can produce. Both modifications result in modifying the class of problems the task can solve. Finally, the modification of the task's conceptual relations modifies the information-processing role of the task, that is, the quality of its output.

Modifying the task's input Let us consider, for example, the situation where a problem solver knows how to plan paths between intersections. Let us also assume that this problem solver is asked to plan a path from one landmark to another. Clearly, the essentials of path planning remain the same, however in order to be able to solve this new problem, the problem solver has to generalize the types of input information it accepts from a pair of intersections to a pair of locations, where a location can be either an intersection or a landmark.

If the modified task gets accomplished by a method, then the subtasks in the task structure that this method produces for the modified task, which consume the same input as the modified task, may have also to be modified to consume the new input of the modified task.

Modifying the task's output Let us consider, once again the above problem solver, and let us assume that it is asked to produce, in addition to the path between the input locations, an estimation of the time it will take to traverse this path. This information is necessary for a scheduler, who assigns errands to agents and uses the estimation of the duration of the path traversal, to infer when each agent will be available for the next errand. In this case, the planner needs to reformulate its "definition" of path planning; it needs to extend the types of information it produces as output to include the information "traversal time".

The planner may already know of "traversal time" as an attribute of the information "path". It can use this knowledge as the defining conceptual relation for a new subtask in its task structure: this new subtask may take as input the path and return as output its traversal-time attribute. The introduction of such a subtask in the planner's task structure extends the types of tasks (information processing functions) that the planner is able to perform.

Modifying the task's conceptual relations In the above path planner, "path planning" is defined as the production of a path such that, its initial node will be the input source location, and its final node will be its input destination location. In a highly connected world, there may be several paths that can fit this conceptual relation. If one wanted to more precisely characterize the paths produced by an individual planner, depending on its particular style, one might specify the conceptual relations of path planning in greater detail. For example, for some planner the path may contain "only highway segments", or for some other planner, the path may contain the "least possible number of turns".

If the task, whose conceptual relations are modified, gets accomplished by a method, then a new subtask may be needed in the task structure that this method produces for the modified task. This new subtask will ensure that the output, as produced by the given method, indeed conforms with the modified conceptual relations of the task to which the method is applied. Note, also, that modifications to a task's input or output may also give rise to the task of modifying the conceptual relations that relate its input to its output.

Modifying a method

Given that a method is defined by the conditions under which it is applicable, the set of subtasks it sets up for the task it decomposes, and the control it imposes over these subtasks, the possible method modifications involve modifying its applicability conditions, the set of these subtasks, or the partial ordering over them.

Modifying the applicability conditions This type of modification is similar to learning operator selection heuristics and may result in improving the efficiency and effectiveness of the problem solving.

Modifying the set of subtasks Modifying the set of subtasks into which a method decomposes a task to which it is applied, modifies the quality of solutions that the overall task produces. The combination of the conceptual relations of the methods subtasks constitutes a multi-step mapping from the overall-task's input to its output. Therefore, modifying the set of the method's subtasks (by including or excluding tasks which perform non-tautological information transformations) is equivalent to modifying the conceptual relations of the overall task. These types of modifications are roughly equivalent to modifying the set of problem-solving operators the problem solver has available to it, by learning new ones, by discarding ones that are evaluated as not useful, or modifying them to perform a slightly different information transformation. Learning macro-operators by aggregating existing operators does not modify the problem-solving mechanism in the same way. It does not change the overall class of information transformations the problem-solving mechanism is able of performing, but rather the efficiency of the process which performs them.

Adding a task Let us return to the example where the problem solver knows how to plan paths between intersections but does not know how to plan paths between landmarks. Let us also assume that the first subtask of the planner's task structure is to locate the

input locations in a map. The planner already knows how to locate intersections in its map; if it could also learn how to locate landmarks in the map, then it could use the rest of its path-planning task structure without change: depending on which type, intersection or landmark, a specific input value is, the planner might use the appropriate task to locate it, and continue planning as previously.

To add a new task in the task structure, the problem solver must discover the conceptual relations that relate this task's input and output. The SBF model explicitly specifies the types of knowledge (world objects and relations) the problem solver has available to it. The problem solver can use this meta-knowledge to discover the relation describing the information transformation desired by the new task. Note, that the introduction of a new task in the task structure, gives rise to information and flow control modifications; the problem solver has to specify when the new task can be performed and which task(s) will consume its output. Again, the SBF model of the problem solving can guide these modifications so that they result in a consistent problem-solving mechanism.

Deleting a task Let us now consider the planner, whose style is to produce paths consisting of highways only (except of the segments entering and exiting the highway system). Let us assume, that to produce this type of paths, the problem-solver's task structure includes the subtask "search for major highways to use": this subtask filters out the non-highway pathways unless when looking for entry and exit points to the highway system. If this planner moves to an environment where such a highway network is not available, the habit of using only highways for planning paths can be overly constraining. As a result the problem solver may start failing to produce paths. Such failures should help the problem solver realize that the "filtering task" is imposing unnecessary, and even worse, unsatisfiable in the new environment constraints. Consequently the problem solver might delete the "filtering task" from its task-structure. Again, the deletion of a task from the task structure, may give rise to information and flow control modifications; for example, the problem solver may to specify alternative types of input for the subtasks which consumed the output of the deleted task.

Modifying the control among the method subtasks In general, the learning tasks of this type result in improvement of the quality of the problem-solving process. Modifications to the control of the problem solving has been addressed by several systems integrating learning and problem solving, as a means to modifying efficiency. At any point during reasoning, several goals may seem relevant to the current state, and several operators may be applicable. Modifications to the control of processing aim to reduce this type of non-determinism, and enable the problem solver make the right choice the first time so that it does not backtrack.

The task-structure framework gives rise to a taxonomy of control–of–processing modifications broader than selection heuristics, such as (a) serialization of parallel tasks, (b) parallelization of serial tasks, (c) reordering of serial tasks, and (d) modifications to the conditions of a subtask within a method. The first type of control modifications can be thought of as the formation of habits regarding the ordering of subtasks which are independent from each other. It results in improved efficiency because it eliminates the deliberation on the relative ordering of these subtasks. The second type can be thought of, as the inverse of the first one: that is relaxation of ordering assumptions which have resulted from a habit-formation process. It results in increased flexibility of the problem-solving process. The third type of modification is usually subservient to the extension of the input of some subsequent task with some type information produced by an earlier task. In this context it results in modification of the quality of the produced solution. Finally, the fourth type of control modifications results in more precise characterization of the role of a subtask within a method, and results in improved efficiency.

Modifying the knowledge

Often the failure of the problem solver to solve a problem, or produce the desired solution for it may be due to its knowledge about the world.

Modifying the content One way of expanding the population of solvable problems, without expanding the class of tasks it knows how to accomplish, is by expanding its world knowledge. Acquisition of new world knowledge leads to increase of the domains in which the problem solver knows how to accomplish its tasks. For example, if the path planner increases its model of the navigation space, i.e. has a model of a larger navigation space than it used to, then it can plan paths for a greater number of pairs of locations, i.e. it can solve more problems than it used to. The need for knowledge acquisition may arise when the feedback of the world contains references to world objects or relations that the problem solver does not know about. In such cases, "parsing" the feedback through its general meta-level understanding of the world can lead to acquisition of new world knowledge.

Modifying the organization As the body of the problem-solver's knowledge increases, the organization of the knowledge becomes all the more critical to the effective use of this knowledge. In order to be able to access the right piece of knowledge when needed, the problem solver has to have an effective knowledge organization scheme. For example, if possible, it is good for the problem solver to have all the conceptual relations that describe the information transformations of its tasks precompiled, instead of having to infer them at run time. Let us consider the "locate the intersection in the map" subtask of our path planner. This subtask takes as input an intersection, and gives as output a neighborhood in which the intersection belongs. If the problem solver does not have a precompiled relation $belongsin : f(intersection) \rightarrow neighborhood$, but only the relation $contains : f(neighborhood) \rightarrow intersection$, then it has to search all the neighborhoods to find the ones(s) in which a specific intersec-

tion belongs. If it had the $belongsin$ relation precompiled, this subtask would have been much more easier to accomplish.

Modifying the representation The choice of a representation for the problem solver's world knowledge may have significant affects to its problem-solving effectiveness (mutilated chess-board problem). Depending on its representation of the world, a problem-solver may or may not be able to represent all the information which can be useful to its problem solving. Modifications to the representation scheme result in improved effectiveness, and increase the potential class of solvable problems. For example, let us assume that our planner represents the intersections as two-street junctions. If it stumbles (feedback from the world or perceptual input) upon an intersection of three streets, and tries to "parse" the new intersection, it may realize that it needs to modify its representation scheme. Changes to the representation scheme may necessitate changes to the information flow in the problem solver's task structure, and may also enable changes to the organization of the knowledge. In our example, when deciding "which way to go next" the planner has more than two options now, and it should take advantage of all of them.

The Reflection Process

Given the range of learning tasks (modifications to the problem-solving mechanism) that the task-structure framework gives rise to, the question becomes, how the problem-solver's comprehension of its own problem-solving mechanism (in terms of a SBF model) enables it to recognize the need for specific modifications. We have developed a process for reflective reasoning, which uses SBF model of the problem-solver's reasoning to

- **monitor** the problem-solver's performance; the SBF model of the problem solver provides a language for interpreting the problem-solver's reasoning steps and generates expectations regarding their outcomes,
- **assign blame** when the problem solver fails; the SBF model of the problem solver along with the trace of the reasoning on a specific problem, enables the localization of the cause of the failure to some element of the problem-solver's task structure, and
- appropriately **redesign** the problem-solving mechanism; the modifications that this subtask performs to the problem solver's reasoning are instances from the taxonomy of learning tasks that the task-structure framework gives rise to. Further, the semantics of the language of SBF models enable the modification of the problem solver in a manner that maintains the consistency of problem solving.

Autognostic [25] is an operational system that implements and evaluates our theory of reflection as a process for "translating" performance failures into operational learning goals, i.e. specific modifications to its problem solving. Autognostic, given the SBF model of a problem solver, monitors its reasoning (subtasks performed, the information produced, and alternative methods considered but not chosen), assigns blame when it fails and it appropriately redesigns its problem-solving mechanism. In this paper, we will sketch out Autognostic's blame-assignment process on top of a multistrategy navigational planner called Router [10, 11].

Router Router's task is to find a path from an initial location to a goal location in a physical space. Router has a topographic model of its world, which contains knowledge about specific world objects (i.e. streets, intersections) and the relations between them (i.e. connectivity). It also has an episodic memory of past problem-solving cases. Router knows two methods for accomplishing the path-planning task: a model-based method, and a case-based method. When presented with a problem, Router chooses one of them based on a set of heuristic rules which evaluate the applicability and utility of the two methods for the given problem.

Autognostic understands Router's problem solving in terms of a SBF model. The SBF model explicitly specifies the two methods that Router can use to plan paths, and the way they decompose the overall problem-solving task in different trees of subtasks, using different types of knowledge. The subtasks resulting from the use of the first method refer to model relations while the subtasks resulting from the use of the second method refer to case-memory relations. Finally, the SBF model contains a meta-level description of the problem-solver's world knowledge (i.e. general descriptions of the different types of objects, such as streets, intersections, and paths, and relations, such as connectivity) and their respective organizations (i.e. general descriptions of the different types of organizational relations, such as organization of the world model in a neighborhood hierarchy, and associations between paths and their initial and final intersections).

Monitoring Problem Solving

As the problem solver solves a particular problem, Autognostic *monitors* its reasoning process and records which method is used for a given task, in which specific order it performs the resulting subtasks, which are the methods invoked for their respective accomplishment and what are their corresponding results. The SBF model of the problem solver provides Autognostic with expectations regarding the information states the problem solver should go through, such as, each information state should be related to the preceding one according to the conceptual relations of the task carrying out the transformation between them, and the completion of a method should signal the accomplishment of the task to which it was applied.

While monitoring the problem-solver's reasoning, Autognostic may realize that some of these expectations fail; Autognostic uses that as an opportunity for learning. If, for example, the invocation of a method fails to accomplish the task at hand although it was applicable to it, Autognostic suggests as possible modification the redefinition of the applicability criteria of the method used, so that it is not used in similar situations in the future. If the cause of the failure is that some of the method's subtasks did not produce

a type of information expected of it, then Autognostic also suggests that this subtask is moved one level higher in the task structure, and that the applicability criteria of the used method are revised to depend on the production of the information type which was not produced in the current problem-solving episode.

Examples When Router's *case-based* method is applied to its *path-planning* task, it sets up as its subtasks *retrieval*, *adaptation* and *storage*. During monitoring of Router's problem solving on the problem of connecting *(10th & center)* to *(8th-3 & peachtree-pl)*, Autognostic notices that *retrieval* fails to find a relevant path in Router's memory, and subsequently, the case-based method does not produce a path for the given problem. Therefore, Autognostic suggests that the *retrieval* subtask is moved before the selection of a method for the path-planning task, and that the applicability of the *case-based* method becomes conditional upon the production of a *middle-path* by the *retrieval* task.

Assigning Blame for the Failure

Even if the problem solver succeeds in producing a solution, there is still the potential of another type of failure: feedback from the environment may inform Autognostic that another solution would have been preferable. This is yet another opportunity for learning.

Autognostic's method for blame-assignment for failures of this type (i.e. production of suboptimal solution) consists of three major subtasks. The first subtask, *assimilation of feedback*, investigates whether the feedback was not produced because it includes references to objects in the world that are unknown to the problem solver or are in conflict with the problem-solver's world knowledge. The second subtask, *blame assignment within the problem-solving strategy used*, investigates whether the strategy used in the failed problem-solving episode, could have produced the feedback under different world-knowledge conditions, or with slight modifications to the way these subtasks were performed. The third subtask, *exploration of alternative strategies*, investigates whether an alternative strategy should have been more appropriate for the problem at hand.

Assimilation of Feedback Information The feedback assimilation process uses the meta-model of the problem-solver's world knowledge to "parse" the desired solution in order to elicit all the information it contains about the world. If some of this information is not part of, or is in conflict with the problem-solver's world knowledge, then this is evidence that the feedback does not belong in the domain of solutions that the problem solver can produce. Then, the assimilation process suggests appropriate modifications to the world knowledge.

The SBF model includes a general description of the types of world objects that the problem solver knows about, including their attributes, the relations that are applicable to them, predicates for testing identity among their instances, and pointers to their domains. For each attribute of a world object, the description includes its type, and a function for inferring its value from an instance. If the instance presented as feedback to the problem solver does not belong in the corresponding world-object domain, the assimilation process adds it to it, and infers its attributes to further examine whether their values are known world objects and whether they agree with their corresponding expected types.

If at any point, the assimilation process faces an inconsistency (failure to infer the value of some attribute, or incompatibility between the value of an attribute and its expected type) between the feedback and the attributes it expects, this is evidence that a change in the representation framework of the problem-solver's world knowledge may be necessary, in order for the assimilation to become possible.

Examples Let us consider the problem of going from *(8th-1 mcmillan)* to *(north cherry-3)*. For this problem Router produces the path *(8th-1 mcmillan) (mcmillan test-3) (test-3 hemphill) (hemphill ferst-2) (ferst-2 ferst-3) (ferst-3 cherry-3) (cherry-3 north)* which is correct but suboptimal to the path *(8th-1 mcmillan) (mcmillan 6th-1) (6th-1 ferst-2) (ferst-2 ferst-3 ponders) (ferst-3 cherry-3) (cherry-3 north)*.

In this example, the assimilation process uses the general description of paths, included in the SBF model of Router's problem solving, to "understand" the feedback. This description specifies a path as a sequence of "nodes" each of which is an intersection. Thus, the assimilation process uses its general description of intersections to assimilate the specific intersections that constitute the feedback path. By investigating the contents of Router's intersection domain, the assimilation process finds that the intersections *(mcmillan 6th-1)* and *(6th-1 ferst-2)* do not belong in it. Moreover, *6th-1* does not belong in the street domain. Therefore, the incorporation of *6th-1* in the street domain, and the incorporation of *(mcmillan 6th-1)* and *(6th-1 ferst-2)* in the intersection domain are suggested as potentially useful modifications.

Finally, while assimilating *(ferst-2 ferst-3 ponders)* as an intersection, the process faces an inconsistency between the type of the attribute *streets* and its value in the specific intersection. That is, while the *streets(intX?)* is a list of two elements, each of which is a street, *(ferst-2 ferst-3 ponders)* is a list of three elements. At this point, the process suggests the modification of the representation of the attribute *streets* of intersections in Router's world knowledge.

Suggesting Modifications within the Strategy used for Problem Solving If the assimilation process proceeds without errors (all the information conveyed by the feedback is already known to the problem solver) the blame-assignment process inspects the strategy used during the failed problem-solving episode, in order to identify modifications which can potentially enable this same strategy to produce the desired solution. It is possible that the desired solution is within the class of solutions that this problem-solving strategy can produce but it was not actually produced because (a) some piece of world knowledge is missing (the successful assimilation of the feedback only means that the constituent elements of the solution

are known to the problem solver; still, there may be types of information used by intermediate problem-solving steps, for which the problem-solver's world knowledge is incomplete), or (b) some type of world knowledge is incorrectly organized, so that although all the necessary information is available to the problem solver, it cannot access it appropriately, or (c) the problem-solving strategy allows the production of several solutions to the same problem, one of which is more desirable than the others.

The blame-assignment process first identifies the highest task in the task structure whose output is the information for which the problem solver produced an undesirable value. It then uses the conceptual relations of the task under inspection to investigate whether it could have produced the feedback value. If the feedback value and the input of this task are verified by its conceptual relations, then the feedback could indeed have been produced by the task under inspection. The blame-assignment method thus infers that the reason why this value was not actually produced must lie within the internal mechanism that accomplished the task under inspection. Thus, the blame-assignment process focuses its attention to the subtask of this which produced the undesired value.

If at some point, the conceptual relations of a task do not hold true between the input of this task and the feedback, then the blame-assignment process attempts to infer alternative input values which would satisfy the failing conceptual relations. If there is some intermediate type of information, for which an alternative value can be inferred, different from the value actually used in the actual problem-solving episode, the focus of the blame-assignment process shifts to identifying why this value was not produced.

The way that the blame-assignment process infers alternative values for the input of the under-inspection task depends on the type of the failing conceptual relation. If the failing conceptual relation is evaluated by a predicate, the inverse predicate may also be known. If it is exhaustively described in an association table, then the inverse mappings can be inferred from this table. Finally, if the input information is a type of world object the instances of which belong in some exhaustively described domain, the blame-assignment process can search the domain to find these values which would satisfy the failing conceptual relation.

If there are no possible alternative values for the the input of the task under inspection that can satisfy its conceptual relations, then the blame-assignment process suggests as possible modifications the following: (a) if the conceptual relations refer to domain relations exhaustively described by truth tables, the updating of the world knowledge to include the mapping of the task's input to its desired output, (b) if the conceptual relations refer to organizational relations, the reorganization of the world knowledge to include the mapping of the task's input to its desired output, or else (c) the redefinition of the task's conceptual relations, so that they don't get violated by the mapping of the task's input to its desired output. The third modification is possible when a new relation can be found, for which the desired mapping is true.

If the blame-assignment process reaches a task which can produce two alternative values, both of them consistent with its conceptual relations, only one of which leads to the feedback solution, this is an indication that the task structure is not sufficiently tailored to producing the right kind of solutions. In this case, the blame-assignment process suggests as possible modifications the following: (a) if the conceptual relations refer to domain relations exhaustively described by truth tables, the updating of the world knowledge to exclude the mapping of the task's input to its old output, (b) if the conceptual relations refer to organizational relations, the reorganization of the world knowledge to exclude the mapping of the task's input to its old output, or (c) the introduction of some new task in the task structure which, using some type of world knowledge, will distinguish between the possible alternatives and will "steer" the problem-solving process towards the correct solution. The third modification is possible when a new relation can be found, applicable to the type of information with the two alternative values, which can differentiate between the actual and the desired value of the task's output.

Examples Let us consider for example, the problem of going from *(10th center)* to *(ferst-1 dalney)*, for which Router produces the path *(center 10th) (10th atlantic) (atlantic ferst-1) (ferst-1 dalney)* which is suboptimal to the path presented as feedback to Autognostic *(center 10th) (10th dalney) (dalney ferst-1)*.

In this example, the blame-assignment process first focuses on the *route-planning* task, as the highest task producing the *path*, and consequently on the *increase-of-path* task, as the subtask of *intrazonal-method*, the method used for *route-planning* when both intersections belong in the same neighborhood. The conceptual relation of *increase-of-path*, *FORALL n in nodes(path) belongs-in(n initial-zone)*, fails for the desired *path* value and the actual *initial-zone* value. The relation *belongs-in* is an organizational relation, and from its truth table, the alternative *initial-zone* value is inferred, *za*. Thus, the blame-assignment process focuses on identifying why *za* was not produced as the value for *initial-zone*. The task producing the *initial-zone* is the task *elaborate*. Its conceptual relations verify both *za* and *z1* as *initial-zone* values. Therefore the blame-assignment process suggests as possible modifications the reorganization of the relation *belongs-in* so that *((10th center) z1)* $\notin$ domain(belongs-in), and the insertion of a new task which will reason about the potential values of *initial-zone* and select the most appropriate one in the context of the given problem.

Exploring Alternative Strategies Often, the desired solution cannot be produced by the same strategy (same sequence of subtasks) that was used during the failed problem-solving episode. Different methods produce solutions of different qualities, and often the desired solution exhibits qualities characteristic of a method other than the one used for problem solving. The blame-assignment process recognizes this "incompatibility" between the desired solution and the method used for problem solving as a suggestion for redefinition of some task's conceptual relations. This is an indication that the feedback is in conflict with

the very definition of some subtask involved in the actual problem-solving. Before revising the definition of this subtask (from now on, we will refer to it as the problem-task) to resolve the conflict, it is worthwhile investigating whether it is possible to pursue another course of reasoning which will avoid it.

The blame-assignment process identifies the last task in the task structure, before the problem-task, for which there exist multiple methods, and which, during problem solving, was accomplished by a method that resulted in the problem-task. We will call this task choice-task. If at the time of method selection, during problem solving, there was another method not chosen yet applicable, this is an indication that this method should have been chosen. Therefore, the blame-assignment process suggests as possible modification the refinement of the method-selection criteria, so that under similar circumstances the alternative method is chosen over the one actually used.

If none of the alternative methods were applicable at the time of method selection, then this is an indication that the problem solver may need to acquire another method. It is possible, however, that a particular alternative method could have produced the desired solution, although it was not applicable. To collect evidence for that potential, the blame-assignment process evaluates the conceptual relations of the tasks arising from the decomposition of the choice-task by each alternative method. If, for some method, there are conceptual relations, which relate the solution information type with types of information available at the time of method selection, that validate the desired solution, then this is evidence that indeed the desired solution fits the "quality" of solutions that this method produces. Therefore, although the method was not applicable at the time, it maybe should have been. Therefore, the blame-assignment process may suggest that the problem solver should try to this alternative method. If its applicability criteria refer to domain or organizational relations, modifications to these relations is necessary before actually testing the method. If it produces the desired solution, the domain/organizational modifications become permanent. If the applicability criterion was a predicate, the blame-assignment process suggests the redefinition of the selection criteria so that the successful method is applicable under similar circumstances in the future.

Examples Let us for example, consider the problem of going from *(fowler 3rd)* to *(fowler 4th)*. Although spatially close, these locations belong in different neighborhoods, and thus Router uses the *interzonal-model-based* method (a model-based method which decomposes the problem into subproblems each of which can be solved within a neighborhood) to solve the problem which results in the path *(fowler 3rd) (3rd techwood) (techwood 5th) (5th fowler) (fowler 4th)*. The desired path should have been *(fowler 3rd) (fowler 4th)*

The within–the–used–strategy blame-assignment step identifies that the feedback path is in conflict with the conceptual relation of the *plan-synthesis* subtask of the *interzonal-model-based* method. This subtask synthesizes the overall path from smaller paths, produced as solutions to the subproblems into which the original problem is decomposed. It takes as input three paths and concatenates them; as a result, the length of the paths it produces is greater than six nodes, which is not true for the feedback path. The blame-assignment process exploring alternative strategies identify that the conceptual relations of *increase-of-path* subtask of the *intrazonal-model-based* method specify that the paths it produces are shorter than six nodes. This is an indication that *intrazonal-model-based* could have produced the path, had it been applicable. Its applicability condition refers to an organizational relation, *belongs-in(final-point initial-zone)*, therefore the blame-assignment process reorganizes the world model to so that *belongs-in((fowler 4th) zd)*. Afterwards, it solves the problem once again. Indeed, the *intrazonal-model-based* is applied and the desired solution is produced. Had it not been the case, it would have undone the modification.

Redesigning the Task Structure

Of course assigning blame is pointless unless it results in the repair of the error and improved problem solving. In this sense, the repair task constitutes one part of the evaluation of any process of blame-assignment. Indeed, in addition to suggested modifications, Autognostic also redesigns the problem solver according to the suggested modifications. The redesign process may be as simple as integrating a new fact in the body of its knowledge about the world, or as complex as introducing a new task in its task structure. In any case, the SBF model of the problem-solver's reasoning task structure guides Autognostic to redesign the problem solver in such a way, that the overall consistency of its behavior will be maintained.

Evaluating the Learning Product

There is no guarantee that the modification will result in an improved problem solver. However, it can be evaluated through subsequent problem solving. If the problem that triggered the modification can now be solved and the appropriate solution produced, this is strong evidence that indeed the modification was appropriate. If not, the problem solver may try other modifications or it may try to evaluate why the modification did not bring the expected results, assuming that it has a model of its reflection process. Reflection, after all, is a reasoning task, and as such it can be modeled in terms of SBF models, just like any other problem-solving task (note, we have been using the language of tasks and methods to describe the reflection process).

Evaluation

Autognostic is a fully operational system. It presently operates in two quite different task domains: on top of Router, in the domain of navigational planning, and on top of Kritik2 [9, 24], in the domain of engineering design. Both Router and Kritik2 are autonomous multistrategy systems developed independently of Autognostic but within the task-structures framework. The widely different tasks that Router and Kritik2 solve suggests that Autognostic's SBF models of problem solving and model-based reflection process are quite general.

We have tested Autognostic quite extensively. In particular, we have tested Autognostic–on–Router for some two dozen problems in each of which Router produced a suboptimal navigation plan, Autognostic–on–Router received the desired plan as feedback, and then Autognostic assigned blamed and suggested modifications. Together the test set of problems covered a number of different kinds of modifications to Router: (i) modifications to Router's world knowledge, for example, the acquisition of the knowledge of streets and intersections, (ii) reorganization of Router's knowledge, for example, modifying the region of space covered by a neighborhood in the system's spatial model of the navigation world, (iii) redesign of Router's task structure, for example, by the insertion of new tasks in its task structure, and (iv) the use of the Router's knowledge in new ways, for example, by using the old knowledge for the newly inserted task. Autognostic–on–Kritik2 identifies errors, by monitoring its performance, in the selection and use of problem-solving strategies (design-modification plans) and suggests the migration of subtasks from one point in the task structure to another for improving strategy selection and application.

Related Research

The issue of integrating learning with problem solving in order to improve performance has received quite a lot of attention in AI. Hacker [22] improves its problem-solving efficiency and effectiveness by acquiring generalized plans and critics from specific plans and failures. Hacker recognizes failures as instances of four types of problematic goal interactions within a plan; thus the critics it can learn are specializations of these four types. Chef [12] employs a similar approach to planning, but it uses specific plans and critics (cases) to construct and criticize its plans. Prodigy [2, 16] is a general problem solver which learns operator selection heuristics. Prodigy and Lex are similar, in that both systems need a complete trace (i.e. a trace which includes several failed efforts of the problem solver and a successful operator sequence) to do blame assignment. The blame-assignment method in Prodigy is more selective than in Lex: it assigns blame or credit to operator applications that fit one of four predefined concepts, where Lex assigns blame to all operator applications that lead outside the correct operator sequence, and credit to all the operator applications within the correct sequence.

Reflection, i.e. reasoning about a problem-solver's behavior, has also been the subject of AI research. An important issue in reflection is whether meta-knowledge and meta-reasoning are fundamentally different than knowledge and reasoning. Wilensky [29] has suggested that the two processes should be the same, and, in a similar spirit, Theo [18] proposes a uniform representation for both knowledge and meta-knowledge. Autognostic's meta-reasoning process is just another reasoning process; note, we have been using the language of tasks and methods to describe it. Also, the knowledge it uses is not fundamentally different from knowledge used in problem solving. In fact, the SBF models of problem solving were adapted by the SBF models of physical devices used in Kritik2 to specify how physical devices work. Max [14] uses reflection only for deliberative integration of its procedures and not for learning. Castle [6] uses a model of the problem solver based on "interacting" components, similar to tasks, to identify undesirable interactions. Castle does not allow for multistrategy reasoning. Meta-Aqua [19] uses a set of special explanation patterns of reasoning failures, Meta-XPs, to identify instances of these errors in the trace of the problem-solver's reasoning.

Functional models (similar to the SBF models of problem solving) have been used to model abstract devices such as programs for software verification [1], students in the context of a tutoring system [13], and knowledge-base validation [27]. Weintraub has developed a blame-assignment method which operates on a task-structure model of the knowledge-based system, to identify errors in the knowledge base. This method is different than Autognostic's in that it relies on knowledge of possible faults for each element of the system instead of a description of its correct performance. Teiresias [5] addresses the same task (knowledge-base validation) in an interactive context.

Concluding Remarks

Traditionally, learning in AI systems has been viewed as a means for improving the effectiveness and the efficiency of the system's problem-solving mechanism. To broaden the scope of learning so that it can also improve the quality of solutions that the system produces, we need to endow intelligent systems with a comprehension of the interactions between the content, the organization and the representation of the knowledge and its use by the problem-solving mechanism.

The SBF model of the problem solving, captures these interactions within the modeling framework of task structures, and enables a reflection process to monitor the problem-solver's performance, and when it fails, to assign blame for its failure to some element of the task structure, (task, method or knowledge) and modify it appropriately. We have shown how this reflection process is able to suggest modifications to

1. the representation scheme of the world knowledge, (modify representation to accommodate discrepancies between feedback and object type)

2. the organization of the world knowledge, (make-relation-true/make-relation-false to organizational relations)

3. the content of the world knowledge itself, (make-relation-true/make-relation-false to domain relations, include new instances to domains)

4. the assumptions on the applicability and utility of different reasoning methods, (revision to method selection criteria)

5. the role of a subtask in the overall reasoning process (insertion of selection tasks, redefinition of a task's conceptual relations)

6. the organization of the subtasks in the task structure (moving one subtask from one point to another)

The above types of modifications result in

1. improving the effectiveness of the system: given a population of problems, the percentage that the system can solve increases as the domain knowledge used by its problem-solving mechanism increases;

2. improving the efficiency of the system: given a set of available methods for some task, the cost of problem solving decreases as the knowledge of the applicability and utility criteria of each method improves; and
3. improving the quality of the overall-task solution: as new tasks are inserted in the task structure and the conceptual relations of existing subtasks are redefined, the quality of the overall problem solution is modified towards meeting new requirements.

References

[1] D. Allemang: Understanding Programs as Devices, PhD Thesis, The Ohio State University (1990)

[2] J. Carbonell: Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In R. Michalski, J. Carbonell and T. Mitchell (eds): Machine Learning: An Artificial Intelligence Approach, Volume II, San Mateo, CA: Morgan Kauffman (1986).

[3] B. Chandrasekaran: Task Structures, Knowledge Acquisition and Machine Learning. Machine Learning 4:341-347 (1989)

[4] W.J. Clancey: Heuristic Classification, Artificial Intelligence 27:289:350 (1985)

[5] R. Davis: Interactive transfer of expertise: Acquisition of new inference rules, Artificial Intelligence 12:121-157 (1977)

[6] M. Freed, B. Krulwich, L. Birnbaum, G. Collins: Reasoning about performance intentions. In Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, 7-12 (1992)

[7] J.S. Gero, H.S. Lee and K.W. Tham: Behaviour: A Link between Function and Structure in Design, In Proceedings IFIP WG5.2 Working Conference on Intelligent CAD, pp. 201-230. (1991)

[8] A. Goel: Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving, PhD Thesis, The Ohio State University (1989)

[9] A. Goel: A Model-Based Approach to Case Adaptation. In Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, Chicago, August 7-10, pp. 143-148 (1991)

[10] A. Goel, T. Callantine, M. Shankar, B. Chandrasekaran: Representation, Organization, and Use of Topographic Models of Physical Spaces for Route Planning. In Proceedings of the Seventh IEEE Conference on AI Applications. 308-314, IEEE Computer Society Press (1991)

[11] A. Goel, T. Callantine: An Experience-Based Approach to Navigational Route Planning. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (1992)

[12] K. Hammond: Case-Based Planning: Viewing Planning as a Memory Task, Boston, MA: Academic Press. (1989)

[13] K. Johnson: Exploiting a Functional Model of Problem Solving for Error Detection in Tutoring, PhD Thesis, The Ohio State University (1993)

[14] D.R. Kuokka: The Deliberative Integration of Planning, Execution, and Learning, Carnegie Mellon, Computer Science, Technical Report CMU-CS-90-135 (1990)

[15] J. McDermott: Preliminary steps toward a taxonomy of problem-solving methods, In Sandra Marcus (ed.): Automating Knowledge Acquisition for Expert Systems, Kluwer Academic Publishers (1988)

[16] S. Minton: Qualitative results concerning the utility of explanation-based learning. Artificial Intelligence 42:363-392 (1990)

[17] T.M. Mitchell, P.E. Utgoff, B. Nudel, R.B. Banerji Learning problem-solving heuristics through practice. In Proceedings of the Seventh International Joint Conference on AI 127-134 (1981)

[18] T. Mitchell, J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette, J. Schlimmer: Theo: A Framework for Self-Improving Systems. In K. VanLehn (ed.): Architectures for Intelligence, Lawrence Erlbaum (1989)

[19] A. Ram, M.T. Cox: Introspective Reasoning Using Meta-Explanations for Multistrategy Learning. In R. S. Michalski and G. Tecuci (eds.): Machine Learning: A Multistrategy Approach IV. Morgan Kaufmann, San Mateo, CA (1994)

[20] Samuel, A., (1959) Some studies in machine learning using the game of checkers, IBM Journal of R&D Reprinted in Computers and Thought, Feigenbaum and Feldman (eds.), (1963).

[21] H.A. Simon: The Sciences of the Artificial. Cambridge, MA: MIT Press, (1981)

[22] J.G. Sussman: A Computational Model of Skill Acquisition, American Elsevier, New York, (1975)

[23] L. Steels: Components of Expertise. AI Magazine 11:30-49 (1990).

[24] E. Stroulia and A. Goel: Generic Teleological Mechanisms and their Use in Case Adaptation. In the Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, pp. 319-324. (1992)

[25] E. Stroulia, A. Goel: Functional Representation and Reasoning for Reflective Systems. Applied Artificial Intelligence: An International Journal (to appear). (1993)

[26] Y. Umeda, H. Takeda, T. Tomiyama, H. Yoshikawa: Function, Behaviour, and Structure. In Gero (ed.): *Applications of Artificial Intelligence in Engineering, vol 1, Design, Proceedings of the Fifth International Conference*, Springer-Verlag, Berlin, pp. 177-193. (1990)

[27] M. Weintraub: An Explanation-Based Approach to Assigning Credit, PhD Thesis, The Ohio State University (1991)

[28] B.J. Wielinga, A.Th. Schreiber, J.A. Breuker: KADS: A modelling approach to knowledge engineering. In Knowledge Acquisition 4(1). Special issue "The KADS approach to knowledge engineering" (1992)

[29] R. Wilensky: Planning and Understanding, A Computational Approach to Human Reasoning. Addison Wesley. (1983)