Learning Problem-Solving Concepts by Reflecting on Problem Solving

Eleni Stroulia and Ashok K. Goel

College of Computing, Georgia Institute of Technology
Atlanta, GA 30332-0280

Abstract. Learning and problem solving are intimately related: problem solving determines the knowledge requirements of the reasoner which learning must fulfill, and learning enables improved problem-solving performance. Different models of problem solving, however, recognize different knowledge needs, and, as a result, set up different learning tasks. Some recent models analyze problem solving in terms of generic tasks, methods, and subtasks. These models require the learning of problem-solving concepts such as new tasks and new task decompositions. We view reflection as a core process for learning these problem-solving concepts. In this paper, we identify the learning issues raised by the task-structure framework of problem solving. We view the problem solver as an abstract device, and represent how it works in terms of a structure-behavior-function model which specifies how the knowledge and reasoning of the problem solver results in the accomplishment of its tasks. We describe how this model enables reflection, and how model-based reflection enables the reasoner to adapt its task structure to produce solutions of better quality. The Autognostic system illustrates this reflection process.

1 Motivation and Background

That which is commonly known as "intelligence" is surely the result of the interaction of a great number of cognitive abilities such as motor control, vision, learning, problem solving, and language use, just to name a few. Yet most AI research does not fully exploit the constraints that these faculties impose on one another. For example, research on problem solving often assumes the existence of rich domain knowledge for solving complex problems but it typically ignores the issue the acquisition of the assumed knowledge. Similarly, research on learning often views the learner as an entity unto itself, and focuses on developing strategies to learn simple concepts without much regard to their usefulness in reasoning. For example, much of learning research has focused on the issue of acquisition of *domain concepts*, e.g., [32, 11, 20]. The task most commonly used for evaluating the products of concept learning has been classification. While the learning of new concepts may expand the range of objects a classification system may recognize, it has little or no effect on the *internal mechanism* of classification. Improving the classification mechanism requires the learning of

problem-solving concepts, i.e. concepts that affect the problem-solving mechanism as opposed to domain concepts which simply refer to classes of objects and relations in the world.

When learning research has aimed towards improving the performance of a problem solver, it has adopted a narrow view of problem-solving. For example, some studies have investigated learning in the context of tasks such as game-playing [27], automatic programming [28], symbolic integration [22], and scheduling [21]. These systems view problem solving as search in a problem space. The search begins at either the initial or the goal state in the problem space, and ends with a sequence of operators that connects the two states. These systems assume the availability of a "complete" set of operators. The only type of information not completely specified is the heuristic rules for selecting the operators. In this framework, the only problem-solving concepts that the problem solver may learn are rules for selecting an appropriate operator to apply at a given state in the problem space. Since these systems can learn only one kind of concept, the impact of learning on problem solving is limited to improving the efficiency of the problem solver. Moreover, for realistically complex tasks, such as planning in a complex environment, it is unreasonable to assume a complete operator set; in such environments, it would be desirable that the system can learn new operators depending on the environmental structure and the task requirements.

Recent work on problem solving has led to a family of theories [6, 33, 19, 5, 29] that describe problem solving at a level higher than that of states, operators, heuristics etc. These theories analyze problem solving in terms of different kinds of generic tasks and generic methods. A task is specified in terms of the kinds of information it takes as input and gives as output. A method is characterized by the kinds of knowledge it uses, and the subtasks it sets up, when applied to some task. Different methods may use different kinds of knowledge, e.g., associative, episodic, or causal knowledge, and set up different subtasks for the same task.

These theories of problem solving use a richer vocabulary of problem-solving concepts. They admit different kinds of concept learning that go beyond learning heuristics for operator selection, for example, learning new tasks and new task decompositions. Note that since these theories posit several different kinds of problem-solving concepts, which play different roles in problem solving, they introduce the issue of what *kind* or kinds of concept to learn in addition to what concept to learn.

Despite the recent popularity of these theories of problem solving, relatively little work has been done on the learning of the problem solving concepts they postulate. We view *reflection* as a core process for recognizing the needs of problem solving, learning the knowledge that can fulfill these needs, and effectively integrating it in the current problem-solving process. We endow the problem solver with a model of its own task structure and with a process capable of monitoring its reasoning on a specific problem, assigning blame, upon failure, to some element in its task structure, and appropriately redesigning its task structure. In this paper, we sketch an architecture for reflective learning, describe a language for specifying the problem-solver's knowledge and reasoning, and discuss

the process for performance-driven reflective learning as a model-based redesign task. We illustrate the learning process using an example from the Autognostic system, a reflective path planner.

2 Reflection for Concept Learning

A Perspective on Problem Solving We adopt Chandrasekaran's [1989] task structures as the framework for analyzing and modeling problem solving. A task consumes some type(s) of information as input and produces as output some other type(s) of information. A task may be accomplished by one or more methods, each of which may decompose the task into a set of simpler subtasks. A method is specified by the kinds of knowledge it uses, the subtasks it sets up, and the control it exercises over the processing of these subtasks. The subtasks into which a method decomposes a task 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.

The tasks and subtasks in the task structure of a problem solver may be instances of generic tasks [4]. A generic task is a task instances of which can be encountered in several domains, such as classification and plan synthesis, and whose methods are applicable to all its instantiations.

A task, in our framework, is specified by the information it takes as input, the information it produces as output, a prototypical task of which it is an instance, and a set of conceptual relations between the input and output information. These conceptual relations constitute a partial description of the correct performance of this task. If the task is accomplished by a method, then the conceptual relations of its subtasks and the ordering relations that the method imposes over these subtasks constitute a partial description of a correct internal mechanism for this task.

The task structure of a problem solver is non-deterministic. Firstly, a task may be accomplished by more than one methods; if more than one method is applicable to a given task in the task structure, then the problem solver opportunistically selects a given method based on some criteria. Secondly, a method may itself be non-deterministic, in that it may specify only a partial ordering for the subtasks it sets up, and some subtasks may not be necessary under specific conditions.

Concept Learning Revisited The task-structure framework of problem solving identifies several problem-solving concepts, and gives rise to corresponding learning tasks, including the following:

1. the criteria for the applicability of a particular method for a given task,
2. the conditions which determine whether and when a subtask of a method is necessary for the progress of the problem solving, and
3. the conceptual relations between the types of information a task takes as input and produces as output.

Thus the products of concept learning may be of several different kinds and may result in different kinds of modifications to the problem-solving process. Learning the applicability criteria of a method to some task is roughly similar to learning heuristic rules for selecting among operators in the problem-solving-as-search framework. Learning the conditions under which to perform a subtask, and learning the conceptual relations between the input and output of a subtask, do not really have equivalents in the problem-solving-as-search framework. If, however, we were to paraphrase them in that framework, then, the former (learning the conditions under which a task needs to be performed in order to contribute to the progress of problem solving) would be roughly equivalent to learning when the application of an operator contributes to the progress towards the goal state, and the latter (learning the conceptual relations that a task imposes between its input and output) would be roughly equivalent to revising the preconditions and post-conditions of an operator or learning a new operator.

3 Reflective Concept Learning

The task-structure view of problem solving compounds the difficulty of concept learning because it raises the issue of deciding "what kind or kinds of concept to learn" in addition to the question of deciding "which concept to learn". In our work, we adopt reflection as the core process for learning problem-solving concepts. We view the reflection process as composed of three abilities:

1. recognizing the need for a new concept of a specific kind in the task structure,
2. identifying the specific concept to be integrated in the task structure, and
3. integrating the new concept so that it results in a valid modified task structure and improved performance.

To enable reflection, we need a well-defined language for representing the task structure of the problem solver. We have adapted the language of structure-behavior-function (SBF) models for describing how physical devices work [12] for this purpose. Adapting the SBF language for modeling how a problem solver works, we express tasks as transitions between information states: the input and output information states of a task describe the types of information that the task takes as input and produces as output correspondingly. Each information-transformation task is annotated by a set of conceptual relations between the task's input and output information, and a set of conditions under which its accomplishment contributes to the progress of problem solving. Moreover, each information transformation is annotated by a pointer to a prototypical task of which it is an instance, and a set of pointers to the methods that can be used to accomplish it. Methods are expressed as partially-ordered sequences of state transformations which specify in detail how they accomplish the task for which they are applicable. Each method in turn is annotated by a set of conditions under which it is applicable to the task. Tasks which are not decomposable by any methods point to the program modules that can directly accomplish them.

The comprehension of a problem solver of its own reasoning in terms of this SBF model enables the problem solver to improve its performance

1. by specifying a "road map" for problem solving which allows the problem solver to *monitor* the progress of its reasoning on a specific problem,
2. by specifying "correctness" criteria for the results of each of the problem-solver's subtasks, so that, when it fails, the problem solver can *assign blame* for its failure to these subtasks whose results are not consistent with their corresponding criteria, and
3. by guiding the problem solver to consistently *redesign* its own problem solving and thus improve its performance.

Monitoring When presented with a new problem, the problem solver uses the model of its reasoning to *monitor* its process for solving this problem. As it solves a given problem, the problem solver records which method it uses for a specific task, which of the resulting subtasks it performs, in which order, by which method, and their corresponding results. The model generates expectations regarding the information states the problem solver goes through as it solves the problem. Each information state in the problem solving should be related to the preceding one according to the conceptual relations of the task carrying out the transformation between them. Also, a task should be performed only when it contributes to the progress of the reasoning.

As the problem solver monitors its reasoning on a given problem, some of these expectations may fail. For example, a conceptual relation of some task may not hold true between the actual values of its input and output information. If, in spite of this failure, the problem solver produces an acceptable solution for the given problem, then the problem solver may recognize the need to modify its understanding of the conceptual relations of task that generated the failed expectations. That is, it may recognize the need to learn a new conceptual relation that can appropriately describe the transformation that this task imposes between its input and output. This is an instance of recognizing the need for learning a type-3 concept.

Another type of expectation failure that may occur during monitoring is that the information produced by some intermediate subtask may not get used by any other subtask. In this case, the problem solver may recognize the need to refine its understanding about the conditions under which the performance of this task contributes to the progress of problem solving. This is an instance of recognizing the need for learning a type-2 concept.

Blame Assignment Even if the problem solving proceeds without expectation failures, the problem solver may produce an incorrect or suboptimal solution. If the problem solver receives the correct solution as feedback from the world, then this too may present another opportunity for learning. In this case, the problem solver can use the record of its failed reasoning process, and the model of its problem solving to *assign blame* for the failure to some element(s) in its task structure and propose modifications which can potentially remedy the problem.

The task-structure view of problem solving gives rise to a taxonomy of learning tasks each one corresponding to a different type of potential cause of failure that the problem solver can identify [30]. In this paper we focus on the learning tasks that involve learning of a new problem-solving concept.

One possible cause for the failure of the problem solver to produce the desired solution to the given problem may be that it did not use the appropriate method to solve it. Often the applicability criteria of some methods may overlap. If the known applicability criteria do not enable the problem solver to discriminate between the methods, then it may choose one arbitrarily. As different methods decompose the overall task into different sets of subtasks, the sets of conceptual relations that describe the transformation of the input problem to the output solution differ from one method to another. Thus, in general, the solutions that each method produces are characterized by different properties. The properties and attributes of the desired solution may suggest that the problem solver should have been used some method, say, $M_{alternative}$, different from the one actually used, say M_{used}. In such a case, the problem solver may recognize that it should modify the applicability criteria of the available methods such that it gives precedence to $M_{alternative}$ over M_{used} in similar situations. This is an instance of recognizing the need for learning a type-1 concept.

Another possible cause for the failure of the problem solver to produce the desired solution may be that as it transforms its input information to produce a solution, it does not "pay attention" to the "right properties" of its problem domain. Often there may be enough information in the problem-solver's knowledge of the world to enable the production of the desired type of solutions. But the problem solver may produce an incorrect solution because it does not make use of the available knowledge. The problem-solver's model of its reasoning and knowledge may lead it to recognize the need for introducing a new subtask in the task structure, such that it uses the knowledge needed for the production of the desired solution type. In this case, the problem solver has to learn the conceptual relations that specify the new task. This is an instance of recognizing the need for learning a type-3 concept.

Redesign After having recognized the need for learning a problem-solving concept of a specific type, the problem solver must employ a learning strategy to actually learn the concept and subsequently *redesign* its task structure to integrate the new concept in it. In the cases of revising the conditions of a method's applicability to a task, or the conditions of a task's usefulness in problem solving, the integration of the new concept does not involve any non-local consequences to the task structure. However, in the case of introducing a new task in the task structure (type-3 concept learning), the consequences to the task structure are non-local. This is because the introduction of a new task implies modifications to the flow of control and information among the existing tasks in the task structure. The semantics of the SBF models can guide the problem solver in its modification process so that the result is a valid task structure as we illustrate below.

Evaluation of Learning There is no a priori guarantee that learning will result in an improved problem solver. However, learning can be evaluated through subsequent problem solving. If the problem that triggered the modification can now be solved and the appropriate solution produced, then 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. The latter assumes a model of the reflection process: reflection is a reasoning task, and it too can be modeled in terms of SBF models just like any other task.

4 Router: A Case-Study Problem Solver

In our work, we use Router, [13], a path planning system, as a case-study problem solver. Router's task is to find a path from an initial location to a goal location in a physical space. Router was not developed specifically for the purposes of this work; thus, originally it did not have a model of its own problem solving. On top of Router, we have developed Autognostic which has a SBF model of Router's reasoning and which is also capable of the reflection process described above. Router and Autognostic together constitute a reflective reasoner and learner.

Figure 1 diagrammatically depicts the architecture we have developed for reflective learning. In this architecture, the reasoner has both reasoning and meta-reasoning capabilities. In Figure 1 tasks are depicted as solid-line, tilted boxes, knowledge is depicted as dashed-line boxes, control flow is shown by double arrows, input and output information flow is depicted by simple arrows, and access and use of knowledge by tasks is depicted by double-headed arrows.

At the reasoning level, Router has domain knowledge. It has a world model which contains knowledge about objects in the world and the relations between them. It also has a case memory which consists of experiences of solving specific problems in the world. Router knows two methods that can achieve this task: a model-based method, and a case-based method. When Router is presented with a problem, it chooses one of these methods based on a set of heuristic rules which evaluate their applicability and utility on the particular problem at hand. At the reasoning level the reasoner, Router, does not have explicit understanding of its knowledge or reasoning.

At the meta-reasoning level, however, the reasoner, Autognostic, understands Router's problem solving in terms of a SBF model. The SBF model of the problem-solver's reasoning captures the interdependencies between the different tasks it can perform, its problem-solving methods, and its knowledge. Autognostic understands that it knows two methods that can be used to achieve the path-planning task. Each one of these methods decomposes the overall problem-solving task in different sets of subtasks, and uses different types of knowledge. One method uses the world model and the other uses the case memory. The problem solver has also explicit knowledge about the ontology on which its world model and case memory are based and their respective organizations. The problem solver uses the knowledge it has at the meta-reasoning level to monitor its

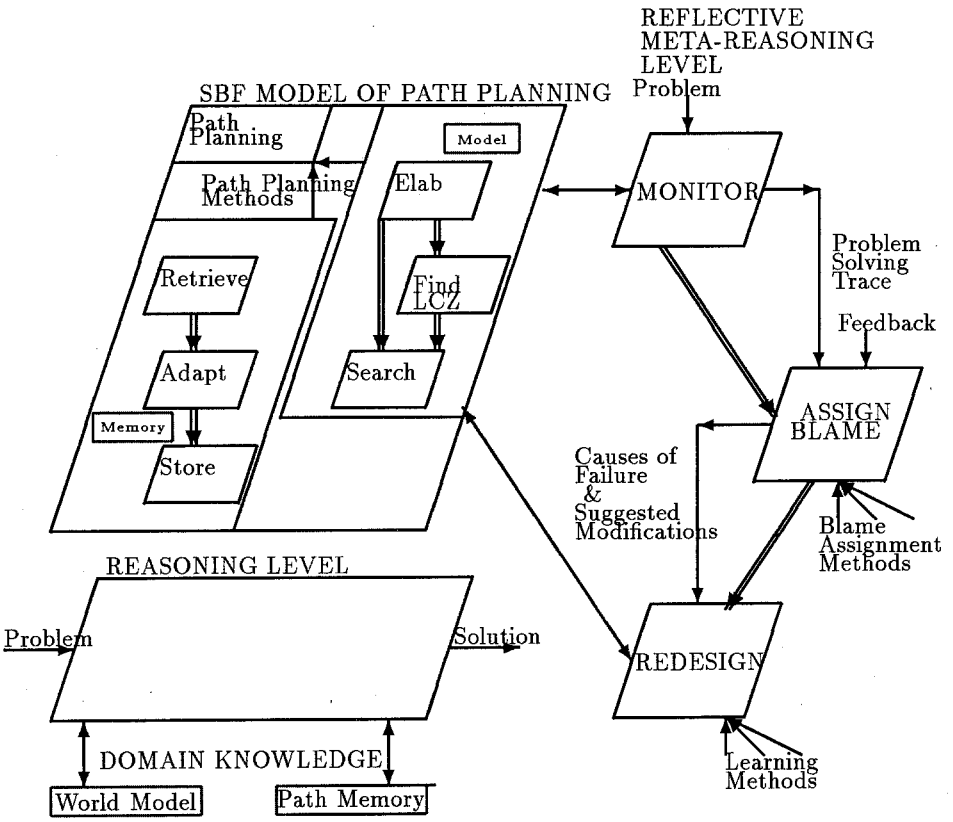

Fig. 1. The Architecture of a Reflective Reasoner and Learner

reasoning, assign blame to some element of its reasoning process it when it fails, redesign it, and thus learn and improve its performance.

Figure 2 depicts a part of the SBF model of Router's problem solving, more specifically a part of its model-based path-planning method. Since the example we discuss in this paper involves the model-based method only, we do not describe in detail Router's case-based method [14]. In the language of SBF models, problem-solving process is viewed as a sequence of transformations between information states. In Figure 2, each information state is depicted as a rectangular box, and contains the information available at the state; each state transformation is depicted by a double arrow, and is annotated by the description of the task which accomplishes the transformation.

Router model of its world is a hierarchically organized topographical model. The model contains knowledge about pathways, their directions and the intersections between them. The pathways are grouped into neighborhoods and the neighborhoods are organized in a space-subspace hierarchy. The higher-level neighborhoods contain knowledge of major pathways, and cover large spaces.

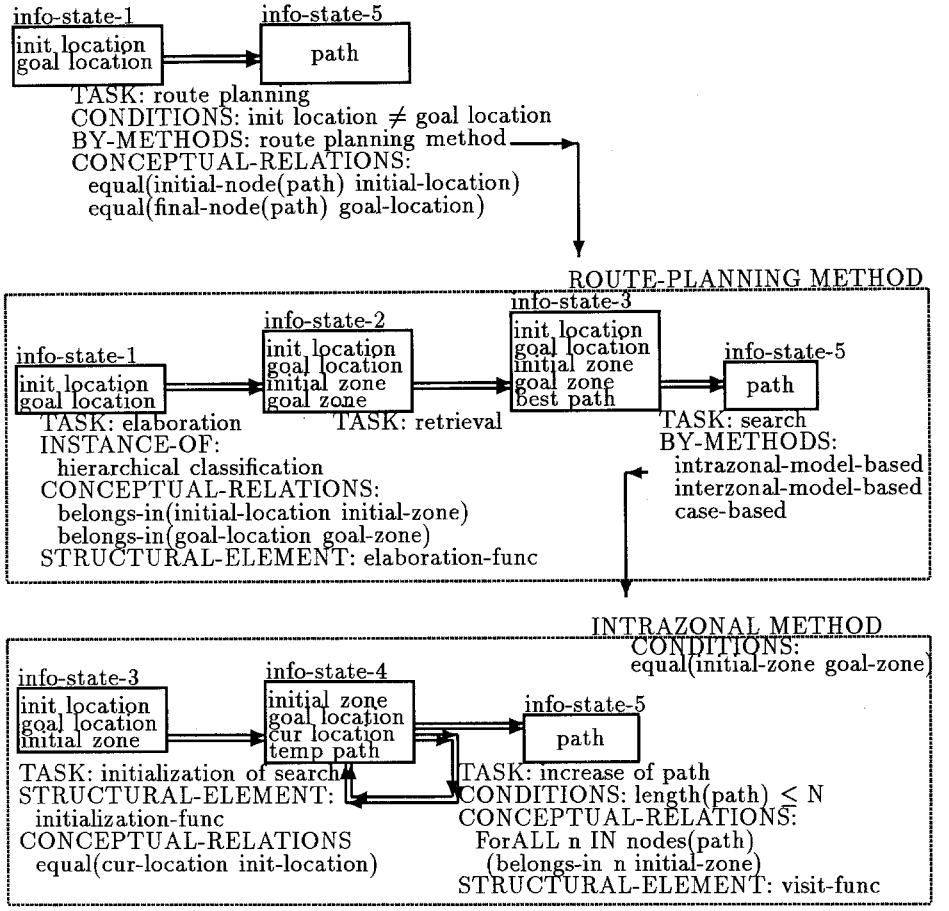

Fig. 2. Part of Router's SBF model

Each neighborhood gets decomposed into several neighborhoods at the immediately lower level. These lower-level neighborhoods contain knowledge of minor pathways but cover spaces smaller than the neighborhood that subsumes them.

When Router is presented with a problem, it first finds the neighborhoods of the initial and goal intersections, `initial-zone` and `goal-zone` correspondingly, with the `elaboration` subtask. Then, the `retrieval` subtask searches in Router's path memory, for a path that is close to the current problem. The conceptual relations of the `retrieval` subtask specify that the retrieved path should connect some intersection in the `initial-zone` to some intersection in the `goal-zone`, to be similar enough to the current problem. If the two problem intersections belong in the same neighborhood, Router may use the `intrazonal-model-based` method to `search` for a path between the two given locations. The `intrazonal-model-based` search method is essentially a breadth-

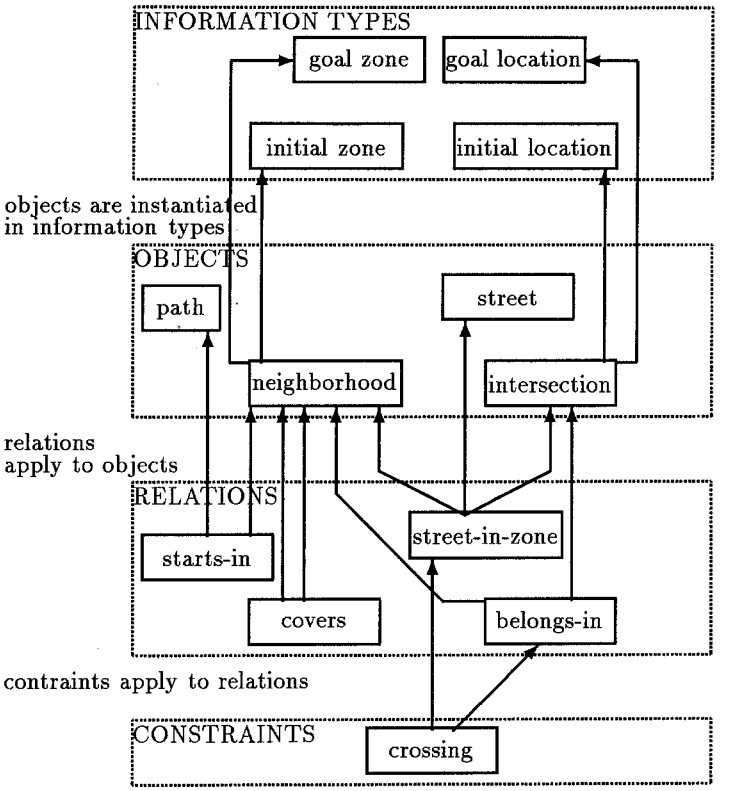

Fig. 3. Fragment of Autognostic's meta-model of Router's world knowledge

first search within the common neighborhood of the two intersections. If the two intersections do not belong in a single neighborhood, Router has two other options for solving the `search` subtask: it can either perform a hierarchical search its neighborhood organization, `interzonal-model-based` method, or it can use the path it retrieved from its memory, as the basis for solving the current problem (`case-based` method). The SBF model of the `intrazonal-model-based` method, is shown in the bottom dashed-line box, in Figure 2. Initially, Router sets up its `current-location` to be the `initial-location`, and initializes its `temporary-path` to contain only this intersection. Then, by repeating the `increase-of-path` subtask, it incrementally adds additional intersections to the `temporary-path`. This subtask is repeated under the condition that the length of the temporary path does not exceed N. If, at some point, Router reaches the `goal-location` Router assigns the value of the `temporary-path` to the `path` and returns it as the desired solution.

Figure 3 depicts part of Autognostic's meta-model of Router's world knowl-

edge. Router's world is described in terms of several different types of **objects**, such as **intersections**, **neighborhoods**, **streets** and **paths**. The **types of information** that Router reasons about in its route-planning process are instances of these objects. The objects in Router's world are related through **relations**, such as **belongs-in** which relates intersections to neighborhoods, and **covers** which relates neighborhoods with other neighborhoods. Finally, some relations are related to each other with domain **constraints**.

5 Learning a Problem-Solving Concept: An Example

In this section we describe in detail Router's reasoning for a specific problem. In this problem, feedback from the world informs Router that there is a better solution to the problem at hand. Thus Autognostic reflects on Router's reasoning using the SBF model of its problem solving as a guide, identifies the cause of the failure and proposes to introduce a new task to Router's task structure. We discuss the adaptation and show how it improves Router's planning performance.

Monitoring the Problem Solving Process Router is presented with the problem of connecting *(10th & center)* with *(dalney & ferst-1)*. Autognostic monitors Router's planning process and generates its trace. The trace is a partial instantiation of the SBF model of Router's problem solving; only the part of the model which explains the subtasks actually performed during the specific problem-solving process gets instantiated. For this example, the trace is the instantiation of the part of the SBF model depicted in Figure 2, where each one of the different types of information is instantiated with the specific values produced during the particular planning session.

Router uses its **route-planning** method to solve the task, and sets up its corresponding subtasks. The **elaboration** subtask produces as output *z1* as the value of both the **initial-zone** and the **goal-zone**. The **retrieval** task returns no path similar enough to the current problem from Router's memory. Since its applicability test, i.e., equality of the initial and goal zones is true, Router chooses to solve the **search** task with **intrazonal-model-based** method. The repeated execution of the **increase-of-path** subtask produces the path *(center 10th East atlantic) (10th atlantic South ferst-1) (atlantic ferst-1 East dalney)* which is returned as the output **path** of the overall **route-planning** task.

Assigning Blame for Producing a Suboptimal Solution The path that Router produced is correct. However it is suboptimal, because it is longer than the path *(center 10th East dalney) (10th dalney South ferst-1)* which is presented to Router as feedback.

Autognostic uses the feedback, the trace of Router's reasoning on the specific problem, and the SBF model of Router's problem solving, to identify the cause of its failure. This model-based method for blame-assignment searches through the task structure of Router's problem solving, and the search is guided by the feedback and the problem-solving trace.

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. In this example, it identifies the `route-planning` task because this is the highest task producing the suboptimal `path`. The process then uses the conceptual relations of the task under inspection to investigate whether the desired value given as feedback could have been produced by the task. If the desired output value and the input of this task are verified by the conceptual relations, then the desired value 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 of the task, that is, it must be due to some of the subtasks which were performed to accomplish the task under inspection. From the trace of the problem solving, the blame-assignment process infers which method was actually used to solve this task, and, as above focuses the assignment of the blame to the subtask which produced the undesired value. In this example, the blame-assignment process focuses initially to `route-planning` and subsequently to `increase-of-path`.

If at some point, the conceptual relations of a task do not hold true between the input of this task and the desired value, then Autognostic tries to infer alternative input values which would satisfy the failing conceptual relations. This is possible when the task's input information is not part of the overall problem specification, in this example { `initial-location goal-location` }, but is produced by some intermediate task in the task structure. Autognostic is able to infer alternative values for the input of the task under inspection in two ways: (a) if the failing conceptual relation is a domain relation exhaustively described in an association table, (Autognostic's meta-level understanding of Router's domain relations, 3, includes a pointer to the data structure holding the relation's association table), then Autognostic can search for the inverse mappings; alternatively (b) Autognostic may know the domain of the desired value (Autognostic's meta-level understanding of Router's domain objects, 3, includes a pointer to the data structure holding the set of the instances of this object known to Router) and it can try to find these values in the domain that would satisfy the conceptual relations of the task. If Autognostic infers an alternative value for some intermediate type of information, the focus of the blame-assignment process shifts to identifying why this value was not produced.

In this example, the conceptual relation of the `increase-of-path` fails for the desired value for `path` and the actual input value for the information `initial-zone`. The relation `belongs-in` is a domain relation, and from its association table, Autognostic infers that the value of `initial-zone` should have been `za`. Thus, the blame-assignment process focuses on identifying why `za` was not produced as the value for `initial-zone`.

If the blame-assignment process reaches a leaf task (that is, a task not further decomposable by a method) whose input is part of the overall problem specification, then there are two possible situations: the desired output value and the actual input values may or may not be consistent with the conceptual relations of that task. In the latter case, Autognostic infers that another method should

probably have been used for the production of the desired value. If there is no alternative method known, then this is an indication that the role of the task under inspection in the task structure, that is its conceptual relations, should be reevaluated.

On the other hand, a leaf task which can produce two alternative values, both of them consistent with its conceptual relations, with one of them however leading to the desired overall problem solution and the other leading to an unacceptable solution, is an indication that the task structure is not sufficiently tailored to producing the right kind of solutions.

ASSIGN-BLAME-SUBOPTIMAL-VALUE(path,
actual value *(center 10th East atlantic)*
(10th atlantic South ferst-1)
(atlantic ferst-1 east dalney)
desired value *(center 10th East dalney)*
(10th dalney South ferst-1))
PRODUCED(path) = { increase of path }
CONCEPTUAL-RELATIONS(increase of path):
ForAll n IN nodes(path) belongs-in(n initial-zone)
Conceptual-Relations hold TRUE for actual values of *path* and *initial-zone*
Conceptual-Relations DO NOT hold TRUE for desired value of *path*
and actual value of *initial-zone*
INFERRING ALTERNATIVE VALUE FOR *initial-zone*
ForAll n IN nodes(desired path) belongs-in(za) ⇒
desired value(initial-zone) = za

ASSIGN-BLAME-SUBOPTIMAL-VALUE(initial-zone,
actual value *z1*
desired value *za*)
PRODUCED(initial-zone) = {elaboration }
CONCEPTUAL-RELATIONS(elaboration):
belongs-in(initial-intersection initial-zone)
Conceptual-Relations hold TRUE for actual values of *initial-zone*
and *initial-intersection*
Conceptual-Relations hold TRUE for desired value of *initial-zone*
and actual value of *initial-intersection*

⟹ `elaboration` can produce either *za* or *z1* for value of `initial-zone`

Fig. 4. Blame Assignment using the SBF model of Router's Path-Planning process

Part of the blame-assignment process for this example is shown in detail in Figure 4. From the desired value of the `path` and the conceptual relation of the producing subtask `increase-of-path`, Autognostic infers that the value of the information `initial-zone` should have been *za*. Both the actual *z1* and

the alternative *za* values of this information meet the conceptual relation of its producing subtask **elaboration**. At this point, Autognostic knows that the **elaboration** subtask can potentially produce either *za* or *z1* as values for the **initial-zone**, because the domain relation **belongs-in(intersection zone)** is not one-to-one.

In this case, Autognostic has two possible modifications actions to choose from: (i) modification to domain relation, that is deletion from the domain relation of the unacceptable mapping, so that it allows only the preferred mapping, and (ii) insertion of selection task, in order to enable the problem solver to select the preferred one mapping when multiple ones are possible. We discuss the first adaptation in [30]; in this paper we describe how the insertion of a selection task affects Router's problem solving.

Redesigning the Problem Solver: Inserting a Selection Task The motivation behind inserting a selection task after the elaboration task in Router's task structure is to enable Router to reason about the two possible values for **initial-zone** and select the most appropriate one. This way, Autognostic can "tailor" Router's task structure towards producing the kind of solutions represented by the feedback.

The SBF model of the problem-solver's reasoning explicitly specifies the ontology of the problem-solver's domain. For each type of information that its tasks consume and produce, the SBF model specifies what type of world object it is. Moreover, for each type of world object, among other things, the model specifies the domain relations which are applicable to it. Autognostic uses this knowledge, along with the specific values (actual and preferred) of the information type to be selected, to discover a relation which can be used to differentiate between these values. If there is such a relation, then Autognostic can use it as a conceptual relation for the new task to be inserted in the task structure.

In our example, Autognostic knows that one domain relation applicable to neighborhoods is the **covers** relation. Given the actual and the alternative values for the **initial-zone**, *z1* and *za* correspondingly, Autognostic notices that *covers(z1 za)*. It then hypothesizes that this can be used as a differentiating criterion between possible alternative values for the **initial-zone**. Thus it inserts in the set of subtasks of the **route-planning** method, after **elaboration**, the **selection-after-elaboration** subtask, with input **intermediate-initial-zone**, (a new information type produced by the elaboration subtask, which does not produce anymore "the" initial zone but all the possible alternatives) output **initial-zone**, and conceptual relation **covers(intermediate-initial-zone initial-zone)**. The new task has as a goal, given a specific path-planning problem, to reason about the possible values of the **initial-zone** in the context of this problem, and select the one which is covered by the rest of them, that is the most specific one.

In more general terms, a newly inserted task in the problem-solver's task structure has as a goal to reason about the possible values of some type of information in the context of a specific problem and select the most appropriate one for the given problem. Thus, the selection-task insertion implies the discovery of

a characteristic property of the information type to be selected which will enable the problem solver to discriminate among the possible values of this information, and select the most appropriate one for a given problem. In our example, selecting the most specific value for the `initial-zone` results in the selection of a low-level neighborhood. Given that lower-level neighborhoods describe smaller spaces in more detail, Router's search becomes very local, and the two problem locations are connected through small pathways instead of major ones, which, in general, results in shorter paths.

In order for the problem solving task structure to be consistent after this modification, Autognostic needs to perform some more modifications in addition to the insertion of the `selection-after-elaboration` task: (i) introduce a new type of information `intermediate-initial-zone` to hold the intermediate results of the `elaboration` task and to be the input of the new task, (ii) create a function to carry out the transformation of the new task, (iii) change (reprogram) the function `elaboration-func` to actually return appropriately a list of values instead of a single one, (iv) modify the description of `initial-zone` in the SBF model to describe as producing task the `selection-after-elaboration` task, and (v) modify the `route-planning` method to include the new task after the `elaboration` task. Autognostic can autonomously perform modifications (i) (ii) and (iv) but not (iii), which is currently performed by a human programmer, at the suggestion of Autognostic.

Evaluating the modified Problem Solver After Router's process is modified, Autognostic evaluates the appropriateness of the revision by presenting Router with the problem that led to failure before. As Router solves the same problem once again Autognostic goes back again to its monitoring task. In our example, Router produces the desired path this time, so the modification can be evaluated as successful. Had Router failed, once again, to deliver the desired path, Autognostic would have another learning opportunity, and it would repeat its blame-assignment-and-learning task.

6 Related and Further Research

Reflection has received much attention in psychological research on metacognition. The main results of this research are that reflection upon own's problem solving enables the problem solver to select particular strategies in particular situations [9], reformulate the course of its own "thinking" to improve performance and meet the varying demands of the task at hand [2], and improve its performance capabilities by monitoring and careful evaluation of its own thinking [17]. Our work on Autognostic is inspired by these results and is consistent with them.

In parallel with research in psychology, AI researchers recognized the usefulness of meta-knowledge, that is knowledge about what they know and how they reason, in intelligent systems [3]. Many AI systems have used descriptions of their own problem solving for several different tasks. These descriptions have

taken a variety of forms depending on the view they adopt for problem-solving and on the task they are used for. Teiresias [7, 8], for example, views problem solving as recursive rule activation. It models its rule base in terms of meta-rules that describe which rules can be used as evidence for or against inferences on domain objects. Teiresias uses its meta-rules to guide the domain expert in identifying erroneous rules in the rule base and acquiring new ones.

Castle and Meta-Aqua are recent systems that use reflection for failure-drive learning. Castle [10] views problem solving as a sequence of interacting components. For each of its components it has a description of its correct performance and a set of intended behavior properties which are important to the effectiveness of this component in the overall architecture. Castle's components are similar to Autognostic's tasks. However, Castle's functional architecture is not hierarchical (a problem solver is a sequence of components) and deterministic (it uses a single sequence of components) which limits its expressiveness. In addition, this limits Castle's blame-assignment task to finding a fault in a single linear component sequence. Moreover, Castle lacks the concept of generic (prototypical) tasks which enables Autognostic to transfer the results of its learning from one point of its task structure to another. Meta-Aqua [26] views understanding as a cycle of explaining its input using XPs, cases and domain knowledge, and modifying its knowledge when anomalies are encountered. It uses a set of special explanation patterns, IMXPs (introspective meta XPs), to parse its trace of reasoning and recognize reasoning failures. Unlike Autognostic, Meta-Aqua does not have explicit descriptions for the "correct" behavior of its reasoning elements (i.e. XP-instantiation, case-interpretation).

NOOS [25] views reasoning as transfer from precedents and uses reflection for learning by memorization of episodes. In NOOS reasoning and learning are modeled in a framework similar to task structures. MAX [18] uses a explicit description of a robot's capabilities to enable deliberative, and consequently more effective, integration of these capabilities. It focuses on self-monitoring rather than recovery from failure.

Failure recovery analysis [15] is another technique for planner modification. It uses statistical analysis of long traces of the planner actions to infer correlation between action patterns and failures. This technique is more appropriate to rapidly changing domains where planning tends to be more reactive to the environment and less deliberative, and where there is no good understanding of the interactions between the different planning strategies.

Functional models, similar to SBF models, have also been used for software program verification [1], knowledge-base validation [31], and student modeling in the context of a tutoring system [16].

Both Router and Autognostic are operational systems. We have evaluated Autognostic for several learning tasks in Router's task domain, for example, the acquisition of new world knowledge, certain kinds of reorganization of the world knowledge, and some kinds of modifications to the task structure along the lines described above. In order to evaluate the generality of Autognostic's language for describing how a problem solver works and its process for reflective

reasoning, we are presently using it in the task domain of engineering design. In the future, we are interested in validating our process-model of reflection against existing psychological data, integrating Autognostic with an autonomous robot and investigating the acquisition of SBF models from source code.

7 Conclusions

We believe that theories of intelligence that artificially divorce learning and problem solving are often under-constrained. Problem solving determines the knowledge needs of the reasoner which learning must fulfill, and learning enables improved problem-solving performance - improvement in problem-solving performance is one way of evaluating the quality of learning. However, different models of problem solving recognize different kinds of knowledge needs, and, as a result, set up different learning tasks and enable different kinds of performance improvement.

The task-structure framework of problem solving gives rise to three different types of problem-solving concepts:

1. the criteria for the applicability of a particular method for a given task,
2. the conditions which determine whether and when a subtask of a method is necessary for the progress of the problem solving, and
3. the conceptual relations between the types of information a task takes as input and produces as output.

We view reflection as a core process for learning these problem-solving concepts. The capability of reflection raises, itself, a set of issues:

1. How to assign the blame for the undesirable properties of the overall problem-solving behavior to some element of the problem-solver's reasoning
2. How to modify elements of the problem-solver's reasoning while maintaining its overall consistency
3. How to represent the elements of the problem-solver's reasoning and the interactions between them, in order to support the previous two tasks, and in such a way, that it is possible to model problem solvers with complex hierarchical, multi-strategy reasoning capabilities

Our work has led us to identify some types of knowledge which must be captured in a framework for modeling problem solving. Such a framework should describe:

1. the subtasks that the problem solver can accomplish and their information needs,
2. the alternative strategies which can potentially accomplish the overall task of the problem solver, and the information and control interactions of the subtasks these strategies consist of
3. the ontology on which the problem-solver's domain knowledge is based

We adapted the language of structure-behavior-function (SBF) models for specifying the functioning of a physical device to develop a modeling framework which satisfies the above knowledge needs. The SBF model enables a reasoner to monitor its problem solving, and, upon failure, to assign blame to some element (subtask or domain knowledge) in its task structure and redesign it appropriately. In this paper, we showed how this process enables the problem solver to

1. recognize the need for a new concept of a specific kind in the task structure,
2. identify the specific concept to be integrated in the task structure, and
3. integrate the new concept so that it results in a valid modified task structure and improved performance.

In addition to improving problem-solving efficiency, which is the general result of learning problem-solving concepts of the first and second kinds, learning concepts of the third kind enables the reasoner to tailor the problem-solving mechanisms to produce solutions of better quality.

Acknowledgements

This work has been supported by the National Science Foundation (research grant IRI-92-10925), the Office of Naval Research (research contract N00014-92-J-1234), and the Advanced Projects Research Agency. In addition, Stroulia's work has been supported by an IBM graduate fellowship.

References

1. D. Allemang: Understanding Programs as Devices, PhD Thesis, The Ohio State University (1990)
2. L. Baker, A.L. Brown: Metacognitive skills of reading. In: D. Pearson (ed.): A Handbook of reading research, New York: Longman (1984)
3. A. Barr: Meta-Knowledge and Cognition. Proceedings of the Sixth International Joint Conference on AI 31:33 (1979)
4. B. Chandrasekaran: Towards a functional architecture for intelligence based on generic information processing tasks. In Proceedings of Tenth International Joint Conference on Artificial Intelligence, 1183-1192, Milan (1987)
5. B. Chandrasekaran: Task Structures, Knowledge Acquisition and Machine Learning. Machine Learning 4:341-347 (1989)
6. W.J. Clancey: Heuristic Classification, Artificial Intelligence 27:289:350 (1985)
7. R. Davis: Interactive transfer of expertise: Acquisition of new inference rules, Artificial Intelligence 12:121-157 (1977)
8. R. Davis: Meta-Rules: Reasoning about Control. Artificial Intelligence 15:179-222 (1980)
9. J.H. Flavell: First discussant's comments: What is memory development the development of? Human Development 14:272-278 (1971)
10. 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)

11. D. Fisher, M. Pazzani: Computational Models of Concept Learning. In D.H. Fisher, M.J. Pazzani, and P. Langley (eds.): Concept Formation: Knowledge and Experience in Unsupervised Learning. Morgan Kaufmann. (1991)
12. A. Goel: Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving, PhD Thesis, The Ohio State University (1989)
13. 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)
14. 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)
15. A. Howe: Analyzing Failure Recovery to Improve Planner Design. In Proceedings of the Tenth National Confrence on AI, 387-392 (1992)
16. K. Johnson: Exploiting a Functional Model of Problem Solving for Error Detection in Tutoring, PhD Thesis, The Ohio State University (1993)
17. R.H. Kluwe: Cognitive Knowledge and Executive Control: Metacognition. In D. R. Griffin (ed.): Animal Mind - Human Mind. Springer-Verlag, Berlin (1982)
18. D.R. Kuokka: The Deliberative Integration of Planning, Execution, and Learning, Carnegie Mellon, Computer Science, Technical Report CMU-CS-90-135 (1990)
19. 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)
20. R.S. Michalski: Inferential learning theory as a basis for multi-strategy adaptive learning. In R.S. Michalski and G. Tecuci (eds.): Proceedings of the First International Workshop on Multistrategy Learning, 3-18, Harpers Ferry, WV (1991)
21. S. Minton: Qualitative results concerning the utility of explanation-based learning. Artificial Intelligence 42:363-392 (1990)
22. 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)
23. 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)
24. J. Piaget: Biology and Knowledge. University of Chicago Press (1971)
25. E. Plaza, J.L. Arcos: Reflection and Analogy in Memory-based Learning. In R. S. Michalski and G. Tecuci (eds.): Proceedings of the Second International Workshop on Multistrategy Learning (1993)
26. 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 (1992)
27. A. Samuel: Some studies in machine learning using the game of checkers. IBM Journal of R&D. (1959) Reprinted in Feigenbaum and Feldman (eds.): Computers and Thought (1963)
28. J.G. Sussman: A Computational Model of Skill Acquisition, American Elsevier, New York, (1975)
29. L. Steels: Components of Expertise. AI Magazine 11:30-49 (1990).
30. E. Stroulia, A. Goel: Functional Representation and Reasoning for Reflective Systems. Applied Artificial Intelligence: An International Journal (to appear). (1993)

31. M. Weintraub: An Explanation-Based Approach to Assigning Credit, PhD Thesis, The Ohio State University (1991)
32. P. Winston: Learning New Principles from Precedents and Exercises. Artificial Intelligence 19 (1982)
33. 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)