Integrating Abstraction, Explanation-based Learning from Multiple Examples, and Hierarchical Clustering with a Performance Component for Planning

Ralph Bergmann
University of Kaiserslautern
W-6750 Kaiserslautern, Germany
E-Mail: bergmann@informatik.uni-kl.de

Abstract

Complex problem solving can be substantially improved by the reuse of experience from previously solved problems. This requires that case libraries of successful problem solutions are transformed into problem solving knowledge with high utility, i.e. knowledge which causes high savings in search time, high application probability and low matching costs in a respective performance component. Planning can be improved by explanation-based learning (EBL) of abstract plans from detailed, successfully solved planning problems. Abstract plans, expressed in well-established terms of the domain, serve as useful problem decompositions which can drastically reduce the planning complexity. Abstractions which are valid for a class of planning cases rather than for a single case, ensure a successful application in a larger spectrum of new situations. The hierarchical organization of the learned shared abstractions causes low matching costs. The presented S-PABS procedure is an EBL-procedure in which abstraction, learning from multiple examples and hierarchical clustering are combined to automatically construct a hierarchy of shared abstract plans by analyzing concrete planning cases. A specific planning procedure has been designed to solve new planning problems guided by the knowledge learned by S-PABS. By allowing a feedback from this planning procedure to the learning component, the integrated system shows an increase in performance through past problem solving.

1 Introduction

Planning improvement can be achieved through the shortening of the problem solving process by the reuse of earlier problem solving experience. While in case-based reasoning [Kolodner, 1987; Riesbeck and Schank, 1989; Althoff and Wess, 1992] a large collection of very detailed cases is organized for efficient retrieval and modification, traditional learning approaches

prefer the extraction of more general knowledge from specific problem solving cases. With explanation-based learning techniques (EBL) [Mitchell *et al.*, 1986] background knowledge about a problem domain can be incorporated to extract control rules [Minton *et al.*, 1989], macro-operators [Fikes *et al.*, 1972; Korf, 1985; Tadepalli, 1991], or skeletal plans [Bergmann, 1992b] as *generalizations* of successful planning cases. Unfortunately, it shows that simply storing generalizations (e.g. as a list of macro-operators) does not guarantee a speed-up effect in general. This is because the sequential search for applicable generalizations and their matching usually exceeds the savings caused by their utilization. This major problem of EBL is called the *utility problem* [Minton, 1990], a problem which has to be seriously considered when building an architecture that integrates EBL-approaches to acquire knowledge for a planning system. In general, the utility of a learned generalization (rule) can be estimated by the following cost/benefit formula also used in the PRODIGY system [Minton *et al.*, 1989] :

$$Utility = (AvrSavings \times ApplicFreq) - AvrMatchCost$$

where $AvrSavings$ is the average time savings produced when the generalization is applicable due to the search elimination, $ApplicFreq$ is the probability that the rule is applicable, and $AvrMatchCost$ is the average time cost of matching the rule.

In the following we want to examine several techniques which can be integratively combined to a new EBL-procedure which analyses previously solved planning cases to learn problem solving knowledge with high utility for new planning problems. Following the cost/benefit formula, rules which lead to high savings ($AvrSavings$) in search time, high application probability ($ApplicFreq$) and low matching costs ($AvrMatchCost$) have a high utility and should be aspired.

A lot of research on planning has intensively examined the savings in search caused by *abstraction* [Sacerdoti, 1974; Friedland and Iwasaki, 1985]. The main computational advantage of having a good abstract solution to a planning problem — an abstract plan — is that one large search space of the complexity b^n (b is the branching factor and n the length of a solution plan) can be decomposed into k subproblems with smaller search spaces, in which the complete problem solution requires a reduced complexity of $b^{n_1} + b^{n_2} + \ldots + b^{n_k}$ ($n_1 + n_2 + \ldots + n_k = n$). As already identified by Knoblock [Knoblock, 1989] it is important to avoid backtracking across subproblems. Fairly independently solvable subproblems should be aspired for which *domain specific abstraction knowledge* is highly demanded. So, learning domain tailored abstractions of successful problem solutions instead of generalizations seems promising since abstractions have shown to lead to significant savings in search time.

The second factor which influences the utility is the application probability of the learned rules. Rules should be applicable for a large spectrum of problems to be solved in the future. Future problems are usually unknown, but can assumed to be similar to the available cases of previous problem solutions. With that assumption, we can aim at learning rules which are applicable for a large set of the previous problems. This leads to approaches for explanation-based *learning from multiple examples* [Flann and Dietterich, 1989].

To reduce the matching costs, Yoo and Fisher [Yoo and Fisher, 1991] have recently pro-

posed the construction of a *hierarchical classification tree* as memory organization structure to store generalizations of examples. A rule stored at a node in such a hierarchy is always more general than the rules stored at any of it's successor nodes. When the match of a rule in the hierarchy fails, all successor nodes can consequently be discarded. So the overall matching costs for the rules can be significantly reduced.

From the combination of the central ideas of these three approaches — abstraction, learning from multiple examples, and hierarchical classification — a learning procedure can be constructed that acquires knowledge, that can be efficiently used to control a performance component for planning. This performance component solves a new planning problem similar to classical hierarchical planning [Sacerdoti, 1974]. The given problem is transformed into an abstract representation first. Then the learned hierarchy is searched for the most specific abstract plan which is applicable, i.e. it matches the abstracted problem description. The problem decomposition imposed by this abstract plan is then used to solve the planning problem on the concrete level.

The rest of this paper presents an approach to automatically transforming a given set of successful planning cases into a hierarchy of abstract plans. In the next section, the five phase S-PABS (Shared Plan Abstraction) procedure is presented as an explanation-based method which learns from a set of concrete planning cases and comes out with shared abstract plans. Section three shows how methods of incremental concept formation can be utilized to construct a classification tree of the learned abstractions. The described learning approaches are demonstrated for the familiar 'Towers of Hanoi' domain. In section four, the performance component — especially it's inference structure — is explained in detail, with the main focus on the way the abstraction hierarchy of plans is utilized. Section five describes how the feedback from problem solving to learning can be established. In a first experiment, the complexity reductions caused by learning from problem solving are investigated. Finally, perspectives and related work is discussed in connection with the presented approach.

2 Explanation-based Learning of Shared Abstractions from Multiple Examples

Michalski and Kodratoff [Michalski and Kodratoff, 1990] have recently pointed out that abstraction has to be distinguished from generalization. While generalization transforms a description along a set-superset dimension, abstraction transforms a description along a level-of-detail dimension which usually involves a change in the representation space from the original into a simpler language [Plaisted, 1981; Tenenberg, 1987; Giordana *et al.*, 1991; Mozetic and Holzbaur, 1991].

In the following, we assume that the two STRIPS-world descriptions (languages) $W_c = (R_c, T_c, Op_c)$ (the concrete world) and $W_a = (R_a, T_a, Op_a)$ (the abstract world) are given, as background knowledge of appropriate descriptions of the planning domain. Each world

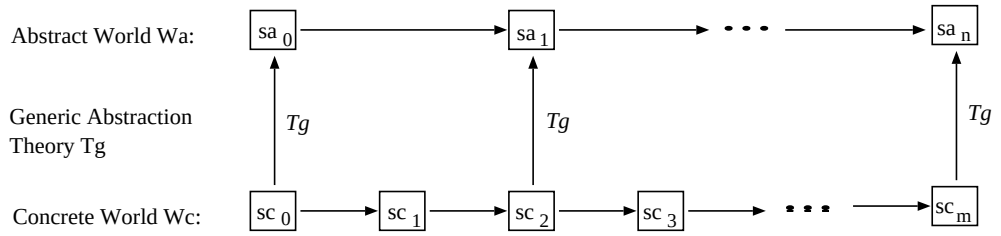

Figure 1: Demonstration of the Plan Abstraction Methodology

description consists of a set of essential sentences R [Lifschitz, 1987] which describe the dynamic aspects of a world state, a static theory T which is assumed to be true in all states of the world, and a set of STRIPS-operators Op specified by precondition, add-list, and delete-list [Fikes *et al.*, 1972]. Furthermore, we assume additional background knowledge which states how the terms of the abstract world , i.e. the essential sentences, can be defined in terms of the concrete world. Generic abstraction theories for semantic abstraction, as introduced by Giordana, Roverso and Saitta [Giordana *et al.*, 1991] relate atomic formulae of an abstract language to terms of a corresponding concrete language. In an adaptation of this idea, a generic state abstraction theory T_g in our model is defined as a set of axioms of the form $\Psi \leftrightarrow D_1 \vee D_2 \vee \ldots \vee D_n$, where Ψ is an essential sentence of the abstract world and $D_1, \ldots, D_n$ are conjunctions of sentences of the concrete world.

A plan is composed of operations which are executed in a specific order and thereby successively change the state of a world. Within this planning model, abstraction has two independent dimensions: On the first dimension a change in the level-of-detail for the representation of single states is described. On the second dimension a change in the level-of-detail is declared by reducing the number of states contained in a plan. As a consequence, a change of the representation of the state description and a change of the operations which describe the state transitions is required. Both dimensions of abstraction are essential to achieve a reduction of the complexity for planning. The problem of plan abstraction can now be described as transforming a plan pc from the concrete world W_c into a plan pa in the abstract world W_a as shown in Figure 1. This transformation must ensure, that all states created by the abstract plan can be derived with the generic abstraction theory T_g from some of the states created by the concrete plan. Thereby, a deductively justified plan abstraction is achieved.

Abstractions which are more useful for reducing planning complexity are those, which are shared by a larger set of concrete plans rather than by a single plan. An abstraction which is shared by a larger class of concrete solution plans and therefore represents a problem decomposition which is applicable for several problem cases, has a higher application probability ($ApplicFreq$) for new problems and is consequently of higher utility. Intuitively, an abstract plan which holds for a set of plans P must be an abstraction for each of the plans. A formal definition of this abstraction methodology can be found in [Bergmann, 1992c] and an extension for shared plan abstractions from several plans is formalized in [Bergmann,

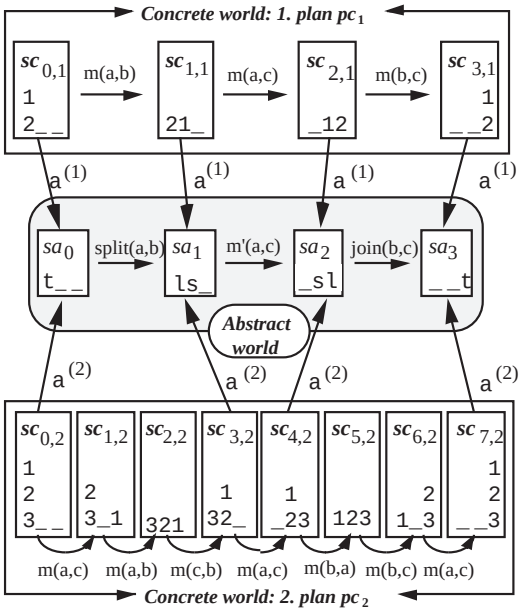

Figure 2: An Example for a Shared Abstraction for two plans

1993].

In the following we want to present a simple example to illustrate the idea of shared plan abstractions.

2.1 An Example of Shared Plan Abstractions

As a planning domain, the familiar Tower-of-Hanoi (ToH) problem — a problem analyzed by several researchers before (e.g. [Korf, 1985]) — is used. The ToH-problem involves three vertical pegs and a number of shaped disks, all of different sizes. In the initial state, all the disks are stacked on one peg in decreasing order of size. The goal is to stack the pegs in the same order on one of the other pegs. The only legal action is to move a single top disk on a peg to another peg subject to the constraint, that a larger disk may never be placed on a smaller disk.

Figure 2 shows two example problem solutions to the 2-disk and the 3-disk ToH problem. In the top of this figure, the plan pc_1 for the 2-disk ToH-problem is shown. In the initial state $sc_{0,1}$ the two disks are stacked on the left most peg a. The other pegs b and c are empty. The stacked disks of the three pegs are represented as three columns of natural numbers, where

the value of a number reflects the sizes of the respective disks (small numbers stand for small disks). An empty peg is represented by the underscore (_) symbol. The first operation of the plan pc_1 is the move of the top disk (1) on peg *a* to peg *b*. This operation is represented by the term *m(a,b)*, which denotes the application of the operator *move* (abbreviated by *m*) with the two parameters source peg *a* and destination peg *b*. The second operation moves the top disk from peg *a* to peg *c* and the third operation achieves the goal state in which all pegs are correctly stored on the right most peg *c*. The bottom of Figure 2 shows the plan pc_2 as a solution to the 3-disk ToH-problem. The three disks (numbered *1,2,3*) are moved with 7 legal moves from peg *a* to peg *c*. The shared abstraction which is derived according to the model of plan abstraction is indicated between the two concrete plans.

Here, the abstract world consists of a different terminology for the descriptions of the states and operations than the concrete world. In the state descriptions, the symbol *t* is introduced as an abstraction of a complete tower of all disks contained in a problem. The symbol *l* stands for the largest disk and *s* abbreviates a small tower of disks, which is a tower that does not contain the largest disk. The generic abstraction theory which is required to allow justified abstraction mappings exactly defines these new abstract objects *(t, l, s)* in terms of the concrete disks. Additionally, new abstract operations which act on the changed state representation are introduced. The operation *split* splits a complete tower *t* into the largest disk and the small remaining tower *s*. The operation *m'* moves a single disk or any tower to a new location. The *join* operation stacks a small tower on top of a large disk. Note that all these abstract operations are not legal elementary operations of the concrete ToH-domain.

2.2 The S-PABS Procedure

In the following, S-PABS (shared plan abstraction) is introduced. This is an EBL-procedure which computes the set of shared abstract plans — denoted as $\mathcal{SA}(P)$ — from a set of concrete plans P, using the given concrete and abstract world descriptions as well as the generic abstraction theory as background knowledge.

Traditional EBL has the property, that the knowledge which is derived from an example as operational generalization is already a consequence of the incorporated domain theory. No knowledge can be acquired which is more general than the theory used for explanation, or more precisely, no rules can be derived which are more general than the rules which are used to *explain* the examples. To allow EBL to come out with described type of plan abstractions, the explanations which are constructed must explain the example plans in terms of the abstract world description. Therefore, our domain theory must not only contain the concrete world description, but also the abstract world description and the generic abstraction theory. Such additional knowledge is not necessary for other approaches for learning generalizations of plans [Minton *et al.*, 1989; Bergmann, 1992b; Fikes *et al.*, 1972].

Moreover, the construction of shared abstractions on possibly different levels of abstraction necessitates the derivation of several alternative explanations (in the abstract world),

for each plan involved. The most specific explanation in the abstract world which is valid for all plans may then be selected as the common explanation. This explanation is complete in the sense, that all operations of the plan are explained, but in a less detailed way. This is in contrast to the kind of shared explanations which are constructed by *Induction over Explanations* IOE [Flann and Dietterich, 1989] or the EXOR system [Yoo and Fisher, 1991]. In these two approaches, the common subexplanation is not anymore complete for all examples.

The proposed S-PABS procedure consists of five distinct phases in which the different types of background knowledge are applied to infer explanations for all concrete plans from which shared abstract plans can be derived. The first three phases are executed separately for each of the plans in P. Phase-IV superimposes the candidate abstract explanations yielding a set of shared plans, which are further variabilized in phase-V. In the following the five phases are explained in detail.

2.2.1 Phase-I: Plan Simulation (Application of Concrete World Knowledge)

By simulating the execution of the concrete plan pc, the state sequence $(sc_1, \ldots, sc_m)$ which is induced by the plan pc and a given initial state sc_0 is computed (see Figure 1). During this simulation, the definition of the operators Op_c and the static theory T_c are applied to derive all those essential sentences which holds in the respective states. The proofs that exist for the applicability of each operator can now be seen as a concrete-level explanation for the effects caused by the operations. Such a kind of explanation is also constructed in EBL-procedures for plan generalizations [Fikes *et al.*, 1972; Chien, 1989; Bergmann, 1992b; Bergmann, 1992a]. For the 3-disk ToH-problem the computed state sequence $sc_{0,2}, \ldots, sc_{7,2}$ is shown in the lower section of Figure 2.

2.2.2 Phase-II: Constructing State Abstractions (Application of the Generic Abstraction Theory)

The second phase performs a prerequisite for composing of the deductively justified abstractions of states. With the generic state abstraction theory T_g, an abstract state description sa'_i is derived for each state sc_i which was computed in the first step. Essential sentences $\Psi \in R_a$ of the abstract world description W_a are checked, whether they can be inferred from $sc_i \cup T_c \cup T_g$. If $sc_i \cup T_c \cup T_g \vdash \Psi$ holds, then Ψ is included into the state abstraction sa'_i. Although, each of the concrete states is transformed with the guidance of the generic abstraction theory into abstract descriptions, not every state has a meaningful abstract interpretation. For which states the abstraction turns out to be useful, can ultimately be answered in phase-IV. For the 3-disk ToH-problem, some of the abstract essential sentences which are derived from the states $sc_{0,2}, sc_{3,2}, sc_{4,2}, sc_{7,2}$ are shown in the center of Figure 2 and also in Figure 3.

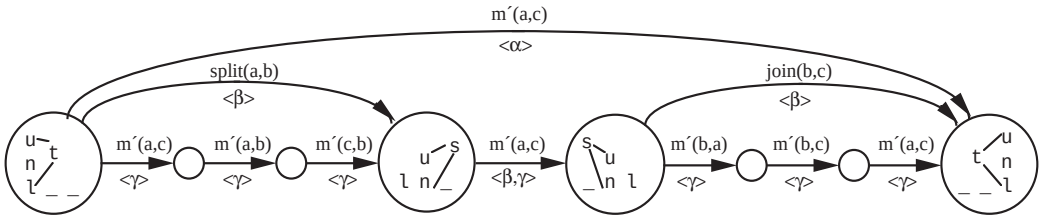

Figure 3: Graph of Candidate Abstract Operations from 3 and 4-disk ToH-problem and Indicated Consistent Paths <α,β,γ>

2.2.3 Phase-III: Constructing Abstract State Transitions (Application of Abstract World Knowledge)

The goal of the third phase is to identify candidate abstract operations for an abstract plan. For each pair of abstracted states (sa'_u, sa'_v) with u < v, it is checked, if there exists an abstract operation $O_\alpha \in Op_a$ described by $< P_\alpha, D_\alpha, A_\alpha >$ which is applicable in sa'_u and which transforms sa'_u into sa'_v. If $sa'_u \cup T_a \vdash P_a$ and if every sentence of A_α is contained in sa'_v and none of the sentences of D_α is contained in sa'_v then the operation O_α is noted to be a candidate for the abstract plan. A directed graph is constructed, where the nodes of the graph are built by the abstract states sa'_i and where links between the states are introduced for those operations that are candidates for achieving the respective state transitions. The proofs that exist for the validation of P_α in sa'_u are stored together with the corresponding operation. A fraction of the complete graph of the candidate abstract operations is shown Figure 3. For some of the abstract states, the description derived in phase-II is shown. In addition to the disk abstraction symbols *t,l,s* which have already been introduced in section 2.1, the symbol n represents the second largest disk and the symbol u represents any tower (complete or partial) which does not contain the largest or the second largest disk. The presented graph can be derived from the 3-disk ToH-problem as well as for the 4-disk ToH-problem.

2.2.4 Phase-IV: Establishing Shared Consistent Paths

From the constructed graph, complete and consistent paths from the initial abstract state sa'_0 to the final state sa'_m are searched, where each path determines a complete abstract explanation. The consistency requirement for such a path pa = $(o_1, \ldots, o_n)(o_i \in Op_a)$ expresses, that every essential sentence which guarantees the applicability of the operator o_{i+1} is created by a preceding operation (through the add-list) and is protected until o_{i+1} is applied, or the essential sentence is already true in the initial state and is protected until o_{i+1} is applied. This condition assures that the plan represented by the path pa is indeed applicable, which means that the preconditions for all operations are satisfied in the states in which they are executed. This consistency requirement can be verified by analyzing the dependencies of the involved operations. The graph shown in Figure 3 consists of five paths

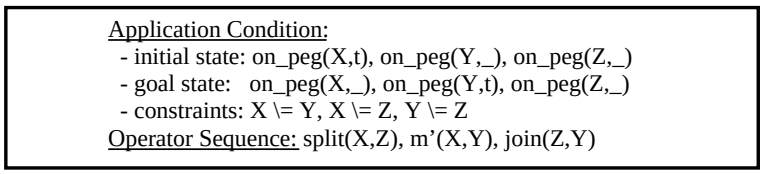

Figure 4: Variabilized Abstract Plan as Result of path $< \beta >$

from the initial to the final abstract state, but only three paths, marked $< \alpha >, < \beta >$, and $< \gamma >$ fulfill the introduced consistency requirement. Although two states (and one operation) are shared by the paths $< \beta >$ and $< \gamma >$, the crossing of them does not represent a consistent path. This is because the operations in path $< \beta >$ work on the disk abstractions noted by the symbols t, l, s, whereas the operations in path $< \gamma >$ rely on a more detailed view on the disk-configuration represented by the symbols l, n, u.

The modules of the S-PABS procedure reported so far, does not take the construction of shared explanations into account. They are executed separately for each of the plans to learn from. The determined set of consistent paths can then easily be superimposed to select only those paths which are shared by all of the example plans. For judging if some paths are shared, it is important to take the intermediate states induced by the plans into consideration too (refer to definition 5).

In the example graph from Figure 3, we can identify that the path $< \alpha >$ is shared by the plans for the 1,2,3 and 4-disk ToH-problem. Path $< \beta >$ represents an explanation for the 2,3 and 4-disk problem, while path $< \gamma >$ only holds for the 3 and the 4-disk problem. From this example we can also see, that possibly more then one path can survive the process of intersection. So all three paths $< \alpha >, < \beta >$, and $< \gamma >$ are shared consistent paths for the 3 and the 4-disk ToH-problem. In this case S-PABS will come out with several plan abstractions from which some may be selected for further usage.

2.2.5 Phase-V: Constructing the Final Abstract Plan Representation

From a shared abstract path and the dependency network which justifies its consistency, a variabilization of the abstracted plans can be established. With the dependency network, which functions as an explanation structure, explanation-based generalization can be applied to compute the least subexplanation which justifies all operations of the abstract path. The proofs that correspond to the justification of the abstract states by the generic abstraction theory are pruned. Within the resulting subexplanation, the remaining derivations are generalized by standard goal regression as used by Mitchell, Keller and Kedar-Cabelli [Mitchell *et al.*, 1986]. Thereby, constants are turned into variables. The final generalized explanation thus only contains relations which describe the generalized operations together with a generalized specification of the application conditions for the operator sequence. As

an example, the variabilized abstract plan which results from path $< \beta >$ is shown in Figure 4. Note that the capital letters X, Z, Y now indicate variables which stand for the pegs of ToH.

3 Classification of Plan Abstractions

This section deals with approaches to an efficient organization and utilization of abstract plans learned by the S-PABS procedure to keep the overall matching costs low. As already proposed by Yoo and Fisher [Yoo and Fisher, 1991], concept formation over explanations is a method that combines the explanation-based paradigm with the paradigm of concept formation [Gennari *et al.*, 1989] to result in a method which can automatically create a hierarchical classification tree of shared explanations.

3.1 Fundamentals for the Hierarchy Construction

The basic idea is to construct a classification hierarchy, in which each node in the hierarchy reflects the shared abstraction of a set of planning cases. The abstract plan stored at a node is valid for all of the node's descendents. A descendent of a node represents a more specific abstract plan, which, when applicable, causes a smaller search space than all of the node predecessors. If an abstract plan at a node is not applicable, than all of the node's descendents will also not be applicable and need not to be visited.

To construct such a classification hierarchy according to the above mentioned requirements, we can easily see that the following property holds for the shared abstractions $\mathcal{SA}(P)$ constructed by S-PABS: $\mathcal{SA}(P1) \subseteq \mathcal{SA}(P2)$ if $P2 \subseteq P1$. This statement expresses the obvious property, that abstractions learned from a set of concrete plans are also valid abstractions for any subset. On the other hand, it is clear that extending the set of plans from which abstractions are to be constructed may reduce the set of resulting shared abstractions. A classification of abstract plans can be build on the basis of a classification of the concrete planning cases. If C_i is a node in the classification hierarchy, then let EC_i denote the set of the concrete example plans which belong to that class. If C_i is a subclass of C_j in the hierarchy, then $EC_i \subseteq EC_j$ holds and therefore $\mathcal{SA}(EC_j) \subseteq \mathcal{SA}(EC_i)$ is also true. An abstract plan pa_{C_i} must be associated with each class C_i, where $pa_{C_i} \in \mathcal{SA}(EC_i)$. A typical representative abstract plan for a class C_i should be chosen in a way that it differs from the abstract plans which have been selected for the superclasses of C_i. To achieve this condition, pa_{C_i} can be designated as follows: $pa_{C_i} \in (\mathcal{SA}(EC_i) \setminus \mathcal{SA}(EC_j))$ if C_j is superclass of C_i. The application domain as well as the classification hierarchy has an important impact on the size of the space of candidates $(\mathcal{SA}(EC_i) \setminus \mathcal{SA}(EC_j))$ from which to select an abstract plan for a class. A perfect hierarchy should be constructed in a way, that for each class the space of candidates contains only one item. In this case the set of abstract plans which is constructed for a set of example plans EC_i is distributed over all the nodes along the path

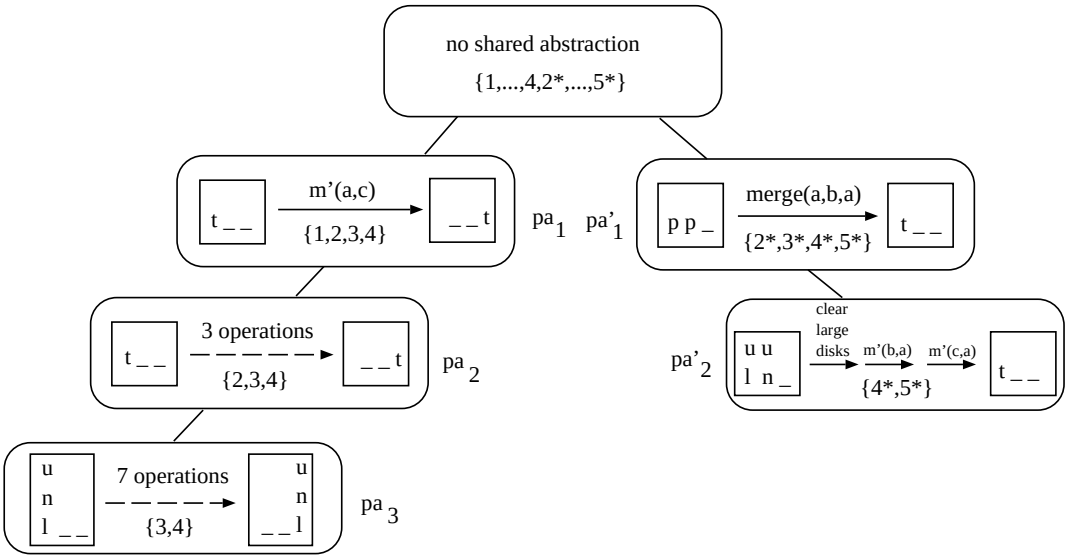

Figure 5: Classification Hierarchy of Plan Abstractions

from the root node of the hierarchy to the class C_i.

An example of such a classification hierarchy for the ToH domain is demonstrated in Figure 5, where 8 different problems are classified. As the first four problems (noted as 1,2,3,4) the traditional 1,2,3 and 4-disk ToH-problems (see section 2.1) are concerned. The other four problems $2^*, 3^*, 4^*, 5^*$ represent a variation of the typical ToH-problems. These problems deal with the merging of two fragmented towers on the pegs a and b into one complete tower located on peg a. In the hierarchy shown in Figure 5, the abstract plans for the different classes and the set of problems from which they are derived by S-PABS are shown. This hierarchy is constructed so, that each class can be characterized by a sole plan since $|\mathcal{SA}(EC_i) \setminus \mathcal{SA}(EC_j)| = 1$. Only the root node of the hierarchy does not contain an abstract plan because no shared abstractions between the traditional ToH-problems and the merge problems exists.

3.2 Incremental Construction of the Hierarchy

For an incremental construction of such a classification hierarchy, a newly observed solution to a planning problem E must be incorporated into an existing hierarchy. For that purpose, S-PABS computes $\mathcal{SA}(EC_i), \mathcal{SA}(\{E\})$ and $\mathcal{SA}(EC_i \cup \{E\})$ for a class C_i of the hierarchy (initially the root). When the three sets of resulting abstractions are compared, three different situations are distinguished:

a) If $\mathcal{SA}(EC_i \cup \{E\}) = \mathcal{SA}(EC_i) = \mathcal{SA}(\{E\})$ then the new example is simply discarded and not incorporated, because all of the examples abstractions are already contained in a class of the hierarchy.

b) If it shows that $\mathcal{SA}(EC_i \cup \{E\}) = \mathcal{SA}(EC_i)$ but $\mathcal{SA}(EC_i) \neq \mathcal{SA}(\{E\})$ then E becomes a member of the class C_i and the classification proceeds to the descendants of C_i. One subclass C_j is chosen in which E fits best. This selection is guided by a criterion that can be rated by $|\mathcal{SA}(EC_j)| - |\mathcal{SA}(EC_j \cup \{E\})|$, the number of abstractions of the class C_j which are not shared by E. (If no subclass can be chosen according to this criterion, then the subclass which contains the largest number of abstract plans is selected). If C_i is a leaf node of the hierarchy, then a new subclass C_j is created, which exactly contains the example plan E.

c) If $\mathcal{SA}(EC_i \cup \{E\}) \neq \mathcal{SA}(EC_i)$ then the new example does not completely fall in the scope class C_i (which is the best selection as guaranteed in case b). Therefore, a new class node called C_k is created and inserted between C_i and its father. This new class is initialized with the example plans of C_i supplemented by the example E ($EC_k := EC_i \cup \{E\}$). Additionally, if $\mathcal{SA}(EC_k) \neq \mathcal{SA}(\{E\})$ a new child of C_k is created, which contains only the example plan E.

With this procedure the classification hierarchy shown in Figure 5 can be incrementally constructed from the eight ToH-problems.

4 The Performance Component

This section describes the performance component that utilizes the hierarchy of abstract plans, constructed with S-PABS, to control the search for solving new planning problems. The hierarchy of abstract plans is traversed and the most specific abstract plan which is applicable is determined. The problem decomposition imposed by this plan is then employed to try to solve the new planning problem. For each of the subproblems created by the application of the abstract plan, a search procedure searchs in the space of the concrete world descriptions for a partial solution plan. Since the applicability of an abstract plan for a given problem does not guarantee that the imposed problem decomposition is successful — even if it has a high application probability — the search for a corresponding solution plan may not be successful. If a certain portion of the concrete-level search space has be unsuccessfully visited, the application of the current abstract plan is interrupted. The next applicable abstract plan is selected from the hierarchy and the given planning problem is tried to get solved by the use of this plan. If none of the applicable abstract plans can be successfully refined to a concrete plan, the concrete-level search space which is visited during plan refinement is enlarged, and the refinement procedure is resumed.

A KADS-like inference structure [Breuker and Wielinga, 1989] of this performance component is shown in Figure 6. As shown on the left side of this figure, the domain knowledge

used as background knowledge for learning (W_c, W_a, and T_g) is also applied for planning. So this knowledge serves two purposes: the knowledge is used to abstract and generalize planning cases and it is used to refine the learned abstractions towards a concrete solution to a new planning problem. This is one aspect of the integration of the learning and the performance component. The basic inferences of this inference structure *state abstraction, hierarchy search, match/instantiation, and controlled concrete-level planning* are now explained in more detail.

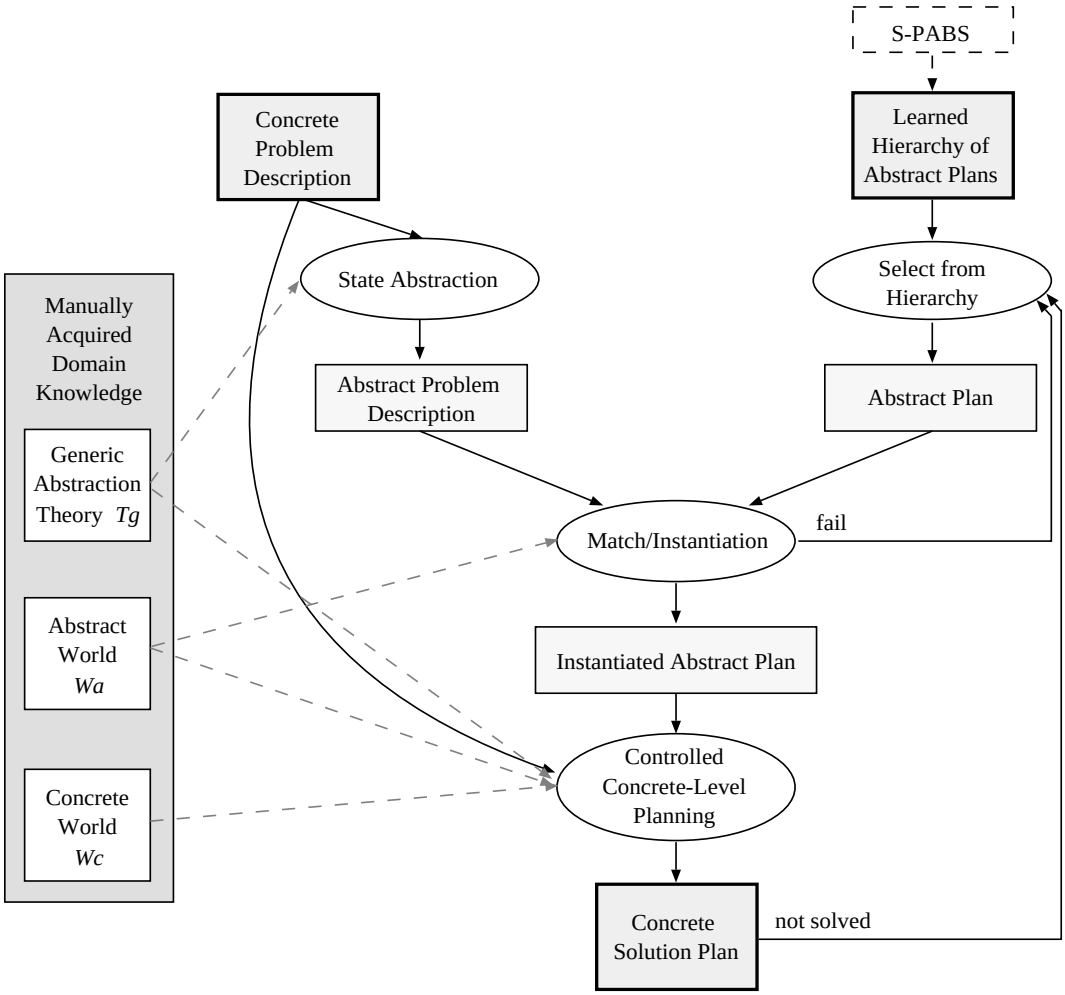

Figure 6: The Inference Structure of the Performance Component for Planning

State Abstraction The basic inference for the state abstraction transforms the concrete problem description consisting of an initial state sc_{init} and a goal state sc_{goal} into an abstract problem description with an abstract initial state sa_{init} and goal state sa_{goal}. Both states

are abstracted independently from each other by the application of the generic abstraction theory T_g as follows: $sc_{init} := \{\Psi \mid sc_{init} \cup T_g \vdash \Psi\}$ $sc_{goal} := \{\Psi \mid sc_{goal} \cup T_g \vdash \Psi\}$

Select from Hierarchy The hierarchy of abstract plans is searched by first selecting the abstract plan at a node of the tree (initially the root node). The match/instantiation procedure is activated and checks, whether the selected abstract plan matches the current abstract problem description. If the match fails, the subtree starting at the current node is abandoned the selection procedure continues by looking at a sibling of the current node. If this match succeeds, the check proceeds recursively to all of the successor nodes of the current node and a more specific abstract plan that matches is searched. If the planning problem cannot be solved with the abstract plans at any successor node, then the concrete-level planning controlled by the abstract plan at the current node is started. If it succeeds, the procedure stops and the problem is solved. Otherwise, the selection procedure continues at a sibling of the current node.

Match/Instantiation The match/instantiation inference is called — as already mentioned — in connection with the select from hierarchy procedure. It's task is to test, if there exists an instantiation of the variables contained in an abstract plan (see Figure 4 as an example), so that the initial state and the goal state of the application condition of the variabilized plan are subsets of the initial state sc_{init} and goal state sc_{goal} of the abstracted problem description, respectively. Additionally, the application constraints in the abstract plan schema must also be satisfied with the current instantiation of the variables. If no such a binding of the abstract plan variables exists, the match fails and a respective feedback is given to the hierarchy selection procedure.

Controlled Concrete-Level Planning This inference procedure solves the given planning problem on the concrete planning level W_c, by constraining the search as defined by the instantiated abstract plan. Thereby, this abstract plan is refined to a concrete solution (cf. [Friedland and Iwasaki, 1985]), which can be seen as the inverse operation of abstraction (see Figure 1). In the first step of this procedure, the sequence of the abstract state descriptions $(sa_0, sa_1, ..., sa_n)$ which corresponds to the instantiated abstract plan is computed. For this computation, the definition of the abstract operators Op_a and the static theory T_a are applied to derive all those essential sentences of the abstract world which holds in the respective states. In the second step, for each state transition from sa_i to sa_{i+1} (starting with $i = 0$) a sequence of concrete-level operations is searched by a deep-bounded breadth-first search in the space of concrete operations. Whenever a node of the concrete search space is expanded by testing a new concrete operation, the resulting concrete state sc_j is determined and it is checked whether all abstract sentences from the aspired abstract state sa_{i+1} can be infered from this concrete state and the generic abstraction theory T_g ($sc_j \cup T_g \vdash \Psi \ \forall \Psi \in sa_{i+1}$). If this is the case, the refinement of this abstract operation was successful, and the next abstract operation yielding sa_{i+2} is handled with the derived concrete state sc_j as the new initial state. If all operations of the abstract plan have been

successfully refined with the outlined search procedure, the last concrete state of the operator chain sc_m is checked, whether it fulfills the concrete goal state sc_{goal} of the planning problem ($sc_{goal} \subseteq sc_m$). If this is the case, the whole planning problem is solved. During the application of this procedure, it can however turn out that the planning problem cannot be solved with the use of the current abstract plan. This is the case if even the search for a sequence of concrete operations for the refinement of a single abstract operation shows to be too complex and cannot be achieved within the current bounds of the search space being examined. In this case, a respective feedback is given to the hierarchy selection procedure.

5 Integration of Learning and Problem Solving

In the previous sections, the integrated learning method S-PABS and the performance component were described separately without concentrating on learning based on the feedback from problem solving.

5.1 Feedback from Problem Solving

In fact, the two proposed components can be used independently from each other: S-PABS can be used as an automatic knowledge acquisition tool that learns from manually solved planning problems and constructs a knowledge base of abstract plans. The performance component can then use this knowledge base to solve newly arising planning problems while the knowledge base remains unchanged. This scenario may be appropriate if we are aiming at a static system for which the spectrum of problems to be solved is known in an advance.

However, the more interesting and more promising systems are those with the ability to adapt themselves over time according to the scope of newly arising problems. To achieve this, a direct feedback from problem solving to learning is required. As shown in Figure 7, this feedback can simply be realized by feeding the learning component with the planning cases solved by the performance component. This can be done due to two properties of the S-PABS: First, its ability to learn from the same data that is produced by the performance component and secondly, by it's incrementality (especially of the hierarchical clustering procedure) which allows it to incorporate newly learned knowledge into the existing hierarchy of abstract plans. This way of learning from problem solving is similar to case-based reasoning [Riesbeck and Schank, 1989; Althoff and Wess, 1992] where adapted cases (as a result from problem solving) are stored in the case base for further reuse. The difference is that in the approach we propose the cases are abstracted before they are (possibly) stored.

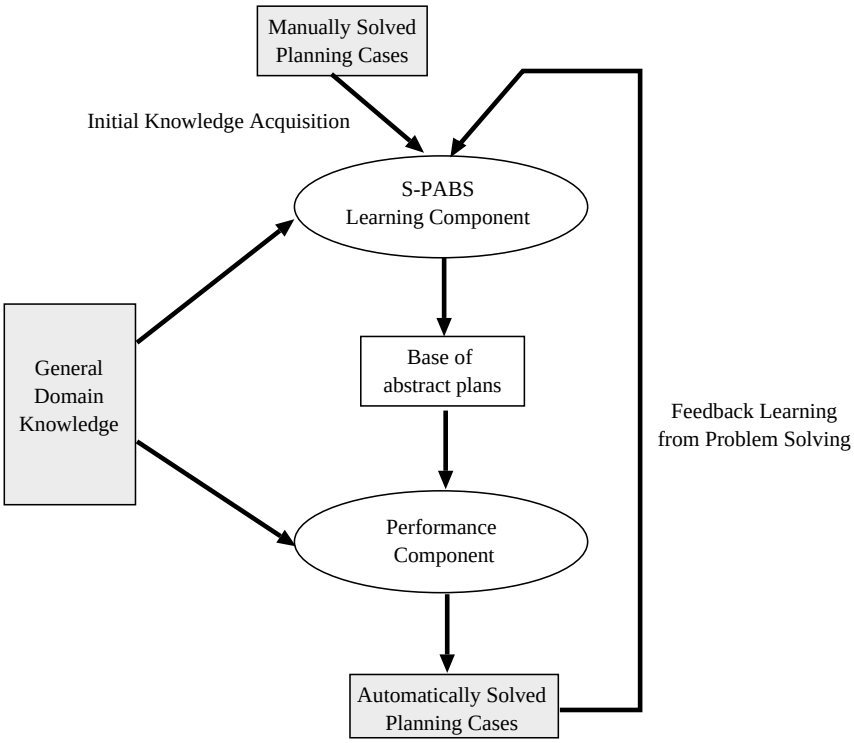

Figure 7: Feedback from Problem Solving to Learning

5.2 Increasing Performance through Feedback

Through learning from problem solving we can expect an increased performance of the system. This is further explained in the following scenario:

In the first step, the performance component is employed to solve a new planning problem. Imagine, that the system shows a bad performance on this problem, i.e. it needs to explore a large search space at the concrete planning level to reach a solution. If this is the case, the "select from hierarchy" inference (see Figure 6) must have choosen an abstract plan (call pa) at a level of abstraction which is too high. So, pa represents a problem decomposition which is too rough for the current problem and consequently causes the performance problem.

In the second step, the feedback from this problem solving process, namely the concrete solution to the planning problem (now called E) is available for S-PABS to learn from. Guided by the incremental clustering algorithm (see section 3) S-PABS produces the abstract plans $SA(\{E\})$ as well as the shared abstract plans $SA(C_i \cup \{E\})$ while it steps through the actual hierarchy. Assuming that the abstract world descriptions contains appropriate abstract operators, an abstract plan pa' is derived from E which is more specific than the

abstract plans already contained in the hierarchy. Therefore, a new class node is created (either in step b) or step c) of the algorithm) which only contains the new example E. Furthermore, the more specific abstract plan pa' is associated with the new class node. Consequently, S-PABS comes out with an updated hierarchy of abstract plans.

In the third step, we assume that the same (or a similar) problem needs to be solved again. In this case, the new abstract plan pa' is available and is selected to solve the current problem. Since pa' is more specific than pa, the search space which must be investigated to refine this plan is smaller than the space searched before learning. This leads to an increased performance of the system for this and similar problems.

5.3 A First Experiment

A first experiment was conducted in the Tower-of-Hanoi domain to test the influence of the learned abstractions on the performance of the system. For this reason, a complexity measure is introduced which counts the numbers of nodes (of the concrete search space) that are examined during planning. The complexity results are shown in Table 1.

Problem	Length of optimal solution	Initial complexity (no learning)	Complexity with incremental learning	Final complexity	Learned abstract plan [1]
2	3	19	19	5	pa_2
3	7	823	36	12	pa_3
4	15	$7.34 * 10^6$	69	69	-
5	31	$2 \ldots 6 * 10^{17}$ (estimation)	6632	6632	-
2^*	1	1	1	1	pa'_1
3^*	3	22	22	22	-
4^*	6	697	697	31	pa'_2
5^*	13	981202	1576	1576	-

Table 1: Comparison of Complexity Results

At first, the system's performance was tested on the previously described 8 ToH- problems (noted as $2, 3, 4, 5, 2^*, 3^*, 4^*, 5^*$) with an empty hierarchy (see "initial complexity" in table 1). All problems are presented (sequentially) to the planning system while the learning component was switched off. So, the abstract plan hierarchy was not updated during this process. It showed that the performance component was able to solve 7 of the 8 planning problems. The solution of problem 4, which required the expansion of $7.34 * 10^6$ nodes, took over 1500 CPU seconds. Problem 5 could not be solved. The estimated search space to solve problem 5 ($2..6 * 10^{17}$ nodes) clearly shows that it cannot be solved by pure search [2].

[1] The abstract plans are referenced as shown in Figure 5

[2] The performance component was implemented in PROLOG on a SPARC-ELC

In the next step, the 8 problems are presented again (in the order listed in table 1) while the S-PABS component is switched on this time. Every time a problem is solved, a new abstract plan is possibly learned and incorporated into the hierarchy. So, an abstract plan learned from the solution of a problem can be immediately utilized to solve the next problem which is presented. The complexity results for the solution of the 8 ToH-problems are presented in the column "complexity with incremental learning" and the abstract plans that are learned after a problem is solved are listed in the right most column of Table 1. It shows, that by presenting the examples ordered by increasing complexity (2..5 and $2^*..5^*$) an abstract plan learned from a problem can be immediately used to drastically reduce the complexity for the next, more complex problem. For example, we can see that the abstract plan pa_3 learned from problem 3 reduces the number of nodes to be expanded for problem 4 from $7.34 * 10^6$ to 69 and even makes problem 5 solvable. This can be seen as an indication that in addition to a performance gain also an increase of competence can result through this learning process.

On the other hand, we can also observe that learning does not appear in all cases. After solving the problems $4, 5, 3^*$ and 5^* no additional abstract plan is learned. The reason for this is, that the abstract world description employed for this experiment does not contain appropriate terms that allow the construction of an abstract plan which is different from the abstract plans that have already been constructed from the previous examples. This leads to the insight that the amount of complexity reduction that can be achieved is dependent on the "vocabulary" of abstract operations available for constructing abstract plans. For the Towers-of-Hanoi domain, this vocabulary could be provided without much effort but for real-world domains similar experiments still have to be conducted.

In the final step of this experiment, the complexity of problem solving is investigated after the complete hierarchy of abstract plans (as shown in Figure 5) has been learned. The final complexity shows an additional speedup since all problems have been solved before and the optimal abstract plans are constructed.

6 Discussion

In this paper, an explanation-based learning procedure was described which integrates abstraction, learning from multiple examples, and hierarchical clustering to learn problem solving knowledge with high utility for planning. A specific performance procedure is designed to solve new planning problems with the use of the learned problem solving knowledge and only the planning knowledge which is already required as background knowledge for learning. Most parts of the proposed learning procedure, as well as the outlined performance component are prototypically implemented in PROLOG. For simple domains — such as the presented Tower-of-Hanoi domain — the learning procedure has shown to lead to the expected performance gains. For real-world domains, which require large libraries of abstract plans, this approach will still has to show how significant the saving in search time will be. Especially, in an experiment with a much larger set of problems to be solved it must be

investigated if the utility problem is still absent.

6.1 Acquisition of the Required Background Knowledge

The most important prerequisite of this method is the availability of the required background knowledge, namely the concrete world description, the abstract world description, and the generic abstraction theory. For the construction of a planning system, the concrete world descriptions must be acquired anyway, since they specify the 'language' of the problem description (essential sentences) and the problem solution (operators). The abstract world and the generic abstraction theory must be additionally acquired. This is indeed the price we have to pay to make planning more tractable. Other hierarchical planning approaches (e.g [Stefik, 1981; Friedland and Iwasaki, 1985; Paulokat and Wess, 1993]) even require the acquisition of multi-level hierarchies of operators. Research on knowledge acquisition has shown that human experts used to employ lot's of abstract knowledge to cope with the complexity of real-world planning problems, e.g in program synthesis [Jeffries *et al.*, 1988; Vorberg and Goebel, 1991] or production planning [Thoben and Schmalhofer, 1990]. Specific knowledge acquisition tools have been developed to comfortably acquire such abstract knowledge [Musen *et al.*, 1987; Bergmann and Schmalhofer, 1991; Schmidt and Zickwolff, 1992] from different sources.

The specific knowledge needs of S-PABS can be fulfilled for the domain of program synthesis of sequential machine-level programs. See [Bergmann, 1992c] for details. In this domain, the concrete world specifies the semantics of the operations of a machine-level programming language. The abstract world represents programming constructs of a higher-level programming language and the generic abstraction theory specifies the abstract data types of the high-level language in terms of the available machine-level data.

6.2 Related Work

Within the Soar framework, Unruh and Rosenbloom [Unruh and Rosenbloom, 1989] have proposed an abstraction technique which can be characterized as general weak method, in that it uses no domain-specific knowledge about how to perform abstractions. This is in contrast to our approach, since we want to draw power from the knowledge about useful domain specific abstractions which have been proven successful in human problem solving.

Unlike other well known techniques for learning search control rules for planning (PRODIGY) by explanation-based learning [Minton *et al.*, 1989], S-PABS can acquire domain oriented problem decompositions rather then more or less restricted operator selection rules. Search control rules can guide the search in a single problem space but cannot reduce planning complexity by switching to an abstract problem description. On the other hand PRODIGY is able to learn from failed solution tracks which actually cannot be performed by S-PABS.

Recently, a few approaches to plan abstraction have been proposed. In Knoblock's method for learning abstract planning spaces [Knoblock, 1989], abstraction always occurs by dropping sentences of the concrete world. This kind of abstraction is only a special case of the type of abstractions we allow.

Similar to S-PABS, the CAbPlan-system [Paulokat and Wess, 1993] — a more traditional case-based reasoning system — also integrates learning from cases with a hierarchical non-linear planner. But planning cases required by CAbPlan are supposed to contain solutions to the planning problem on all levels of abstractions and not only on the concrete level as S-PABS does. Furthermore, CAbPlan assumes a sole abstraction mapping (for states) which is valid for all planning problems, while S-PABS constructs a deductively justified abstraction mapping for each problem. So, S-PABS seems to be an approach with displays it's strengths in domains, where various different ways of abstracting problems and solutions have to be recognized and utilized, while CAbPlan is appropriate for domains, where a uniform kind of abstraction for all problems to be solved can be established.

Acknowledgements

This research was partially funded by the Commission of the European Communities (ESPRIT contract P6322, the INRECA project). The partners of INRECA are AcknoSoft (prime contractor, France), tecInno (Germany), Irish Medical Systems (Ireland) and the University of Kaiserslautern (Germany). I would like to thank Dagmar Surmann for implementing a prototypical version of the performance component, and Wolfgang Wilke for the implementation of PABS. Thanks also to Prof. Richter and to our research group at Kaiserslautern to make this work possible.

References

[Althoff and Wess, 1992] K.D. Althoff and S. Wess. Cased-based reasoning and expert system development. In F. Schmalhofer, G. Strube, and Th. Wetter, editors, *Contemporary Knowledge Engineering and Cognition*, pages 146–158, Heidelberg, 1992. Springer.

[Bergmann and Schmalhofer, 1991] R. Bergmann and F. Schmalhofer. Cecos: A case experience combination system for knowledge acquisition for expert systems. *Behavior Research Methods, Instruments and Computers*, 23:142–148, 1991.

[Bergmann, 1992a] R. Bergmann. Explanation-based learning for the automated reuse of programs. In *Proceedings of the IEEE-Conference on Computer Systems and Software Engineering, COMPEURO92*, pages 109–110, 1992.

[Bergmann, 1992b] R. Bergmann. Knowledge acquisition by generating skeletal plans. In F. Schmalhofer, G. Strube, and Th. Wetter, editors, *Contemporary Knowledge Engineering and Cognition*, pages 125–133, Heidelberg, 1992. Springer.

[Bergmann, 1992c] R. Bergmann. Learning plan abstractions. In H.J. Ohlbach, editor, *GWAI-92 16th German Workshop on Artificial Intelligence*, volume 671 of *Springer Lecture Notes on AI*, pages 187–198, 1992.

[Bergmann, 1993] R. Bergmann. Learning hierarchically clustered shared plan abstractions as problem solving knowledge with high utility for planning. In A. Horz, editor, *Proceedings of GI-Workshop on 'Planung und Konfigurierung' PuK*, number 723 in Arbeitspapiere der GMD, pages 97–108, Bonn, 1993. GMD.

[Breuker and Wielinga, 1989] J. Breuker and B. Wielinga. Models of expertise in knowledge acquisition. In G. Guida and C. Tasso, editors, *Topics in expert system design, methodologies, and tools*, pages 265–295. North Holland Publishing Company, Amsterdam, The Netherlands, 1989.

[Chien, 1989] S. A. Chien. Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of the International Joint Conference on Artificial Intelligence-89*, volume 1, pages 590–595, Detroit, MI, 1989. Morgan Kaufmann.

[Fikes *et al.*, 1972] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Flann and Dietterich, 1989] N.S. Flann and T.G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4(2):187–226, 1989.

[Friedland and Iwasaki, 1985] P. E. Friedland and Y. Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, pages 161–208, 1985.

[Gennari *et al.*, 1989] J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[Giordana *et al.*, 1991] A. Giordana, D. Roverso, and L. Saitta. Abstracting background knowledge for concept learning. In Y. Kodratoff, editor, *Lecture Notes in Artificial Intelligence: Machine Learning-EWSL-91*, pages 1–13, Berlin, 1991. Springer.

[Jeffries *et al.*, 1988] R. Jeffries, A. A. Turner, and P. G. Polson. The processes involved in designing software. In J. R. Anderson, editor, *Cognitive Skills an their Acquisition*, pages 255–283. Lawrence Erlbaum Associates, Hillsdale, NJ, 1988.

[Knoblock, 1989] C. A. Knoblock. A theory of abstraction for hierachical planning. In *Proceedings of the Workshop on Change of Representation and Inductive Bias*, pages 81–104, Boston, MA, 1989. Kluwer.

[Kolodner, 1987] J. L. Kolodner. Extending problem solver capabilities through case-based inference. In Morgan Kaufmann, editor, *Proceedings of the 4th International Workshop on Machine Learning*, pages 167–178, Irvine, Ca, 1987.

[Korf, 1985] R. E. Korf. Macro-operators: A weak method for learning. *Artifical Intelligence*, 26:35–77, 1985.

[Lifschitz, 1987] V. Lifschitz. On the semantics of strips. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 1–9, Timberline, Oregon, 1987.

[Michalski and Kodratoff, 1990] R. S. Michalski and Y. Kodratoff. Research in machine learning: Recent progress, classification of methods, and future directions. In Y. Kodratoff and R. S. Michalski, editors, *Machine learning: An artificial intelligence approach*, volume 3, chapter 1, pages 3–30. Morgan Kaufmann, San Mateo, CA, 1990.

[Minton *et al.*, 1989] S. Minton, J. G. Carbonell, C.A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning:a problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[Minton, 1990] S. Minton. Quantitativ results concerning the utility of explanation-based learning. *Artifical Intelligence*, 42:363–391, 1990.

[Mitchell *et al.*, 1986] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[Mozetic and Holzbaur, 1991] I. Mozetic and C. Holzbaur. Extending explanation-based generalization by abstraction operators. In Y. Kodratoff, editor, *Lecture Notes in Artificial Intelligence: Machine Learning-EWSL-91*, pages 282–297, Berlin, 1991. Springer.

[Musen *et al.*, 1987] M. Musen, L.M. Fagan, D.M. Combs, and E.H. Shortliffe. Use of a domain model to drive an interactive knowledge-editing tool. *Int. J. Man-Machine Studies*, 26:105–121, 1987.

[Paulokat and Wess, 1993] J. Paulokat and S. Wess. Cabplan - fallauswahl und fallbasierte steuerung bei der nichtlinearen, hierarchischen planung. In *(This volume)*, 1993.

[Plaisted, 1981] D. Plaisted. Theorem proving with abstraction. *Artifical Intelligence*, 16:47–108, 1981.

[Riesbeck and Schank, 1989] C.K. Riesbeck and R.S. Schank. *Inside case-based reasoning.* Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1989.

[Sacerdoti, 1974] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Schmidt and Zickwolff, 1992] G. Schmidt and M. Zickwolff. Cases, models and integrated knowledge acquisition to formalize operators in manufacturing. In *Proceedings of the 7th Knowledge Acquisition for Knowledge-based Systems Workshop (Banff)*, 1992.

[Stefik, 1981] M. Stefik. Planning and meta-planning(molgen: part 2). *Artificial Intelligence*, 16:141–170, 1981.

[Tadepalli, 1991] P. Tadepalli. A formalization of explanation-based macro-operator learning. In Morgan Kaufmann, editor, *Proceedings of the International Join Conference on Artifical Intelligence-91*, pages 616–622, 1991.

[Tenenberg, 1987] J. Tenenberg. Preserving consistency across abstraction mappings. In J. McDermott, editor, *Proceedings of the 10th International Conference on Artifical Intelligence*, pages 1011–1014, Los Altos, CA, 1987. Morgan Kaufmann.

[Thoben and Schmalhofer, 1990] J. Thoben and F. Schmalhofer. Wiederholungs- varianten- und neuplanung bei der fertigung rotationssymmetrischer teile. Technical report, DFKI, Kaiserslautern, 1990.

[Unruh and Rosenbloom, 1989] A. Unruh and P.S Rosenbloom. Abstraction in problem solving and learning. In *Proceedings of the International Join Conference on Artifical Intelligence-89*, pages 590–595, Detroit, MI, 1989. Morgan Kaufmann.

[Vorberg and Goebel, 1991] D. Vorberg and R. Goebel. Das lösen rekursiver programmierprobleme: Rekursionsschemata. *Kognitionswissenschaft*, 1:83–95, 1991.

[Yoo and Fisher, 1991] J. Yoo and D. Fisher. Concept formation over explanations and problem-solving experience. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence*, volume 2, pages 630–637, San Mateo, CA, 1991. Morgan Kaufmann.