Generalized Planning and Information Retrieval

Michael M. Richter

Generalized Planning and Information Retrieval

Michael M. Richter
University of Kaiserslautern
Center for Learning Systems and Applications (LSA)
PO-Box 3049
67653 Kaiserslautern, Germany
Email: richter@informatik.uni-kl.de

I. Introduction

For diagnostic processes it has long been accepted that the establishment of the diagnosis is only the last step in a diagnostic process. Most of the work is devoted to the (partial completion) of an incomplete state of information about the world. This world is e.g. the health status of a patient or the failure status of a machine. Often the world is described by an attribute-value representation and in this case the strategy of the diagnostic process has to select attributes for which missing values have to be acquired. This acquisition is connected with costs and hence the diagnostic process includes an optimization problem. On the other hand, the therapy is usually separated from the diagnostic process; the process finishes with the establishment of the diagnosis. Hence the whole diagnostic process gives rise to a specific planning problem. The purpose of this plan is to obtain the needed information in an efficient way. One can view this as planned information retrieval.

The most common model for AI planning is based on the STRIPS notation. Certain actions are defined in terms of operators, preconditions, add and delete lists. The problem is to find a sequence of operator applications which transfer an initial situation into some desired goal situation. In this article we assume the view of systematic nonlinear planning (SNLP), see e.g. [McAllester 91].

It was mostly assumed that the knowledge about the world is sufficiently complete for building the plan. Therefore there is no need for gathering new information about the world; the actions are carried out in a completely known and fully deterministic environment. Recently this view was broadened, cf. [Pryor 95]), but to our knowledge this was not realized in practical systems.

The needed information is often distributed over different locations. It is typical for information retrieval that

a) the content of the information sources is not precisely known;

b) several sources have to be accessed.

In addition, different from database retrieval, one often does not know exactly what one is looking for and this becomes only clear during the search. For these reasons, information retrieval has also to be planned The goals of such a plan have been named as *knowledge goals* (cf. [Pryor 95]). In order to achieve a knowledge goal the plan has to produce a number of (intermediate) pieces of information. These knowledge pieces do not only have a truth value but are in addition more less useful (which is finally the only thing that matters). This is standard when one is building decision trees and is also accepted in diagnostic processes.

Another aspect of many planning problems is that certain actions have to be carried out before the plan is finally established and even before the planning situation is completely known. This requires an interleaving of planning and execution of the plan. This is usually the case when large projects are planned. In addition, during planning as well as during execution one has to react on changes of the environment which is often not stable. As a consequence, some truth maintenance system is required.

We will also discuss shortly which impact these considerations have on case-based reasoning (CBR). We assume that the reader is familiar with the basic terminology and techniques of CBR, cf. e.g. [Wess 94]

In this note we will present a view in which these aspects, in particular both diagnosis and planning are embedded in more general and unified setting. The terminology will be kept as simple as possible. The intended benefit is to reuse techniques which have been developed for some purpose in a different context. For this we refer to several systems designed at the university of Kaiserslautern.

II. The general problem situation

The situations we are interested in may have many aspects which all occur in real life applications, although not always simultaneously. We will first list the main characteristics of such aspects:

(1) The world is only partially known.

(2) Knowledge about the world may be uncertain or even false.

(3) There are actions which change the world, actions which change the knowledge about the world, i.e. information providing actions and actions that do both.
(4) Actions can but may not necessarily be executed.
(5) The different actions interact with each other.
(6) The execution of actions may be forced.
(7) There may be unexpected and unavoidable actions executed.
(8) Actions have costs and benefits.

Next we will indicate some situations where these difficulties occur. This is always the case in general decision making with an open world. More specific situations are:
(1) This is first the standard situation in diagnosis; the main task in a diagnostic process is to partially complete the information such that appropriate further actions can be performed. Also in robotics the robot never has a complete view of the world. Other examples are planning or configuration tasks of complex products. The initial information is rarely sufficient and often up to modifications.
(2) The outcome of many technical processes (e.g. in chemistry) is often uncertain. Certain results of an action may depend on the weather of tomorrow. Due to errors information about regulations may be false.
(3) If one travels from A to B the state of the world is changed and if one looks up a train schedule then the information is changed (i.e. new information is added). Walking to the train station and looking up the schedule changes both, the state of the world and the state of the information; in addition, there might be the unexpected information that there is also a bus going to the point of destination.
(4) A telephone call may be unsuccessful, a machine may not operate.
(5) The interaction STRIPS-like actions is usually described by causal links, cf. [ ]. When information providing actions are present new aspects become involved:
(a) Certain actions can only be planned if another action is executed, e.g. travel plans depend on the knowledge of available transportation means.
(b) The failure of the execution of an action gives the information that some condition is not satisfied (e.g. one has dialed the wrong telephone number or an engine has no fuel). The same is the case when the outcome of an executed action is not as expected.
(6) Some authority (e.g. a legal one or a customer) may force the immediate execution of an action. Also, if due to lack of knowledge no meaningful actions can be planned an execution may be forced or some information providing action has to be done.

(7) A telephone call may be interrupted or it starts raining.
(8) In addition to the costs and benefits one has here to take into account the benefit of the obtained information as well as the possible costs of using an unreliable information.

One observes immediately that several points mentioned above enter the scenario only because actions are not only planned but in addition executed. Therefore we distinguish strongly between an action which is planned and an action which is executed. The difference is mainly that there is no backtracking possible for executed actions and their costs cannot be removed.
In order to distinguish planned actions from executed ones we will use the term "postcondition" for planned actions and use the term "effects" for executed actions.
In addition, an executed action may have unforeseen effects which contribute to our knowledge about world. In terms of ordinary programming the distinction can also be phrased as follows:
- Planning takes place at compile time;
- execution takes place at run time.
In this view ordinary planning is related to writing a program which is afterwards executed while the interleaving of planning and execution is related to the interpretation of programs. It is suitable to have the following identifications:

planning time = compile time
execution time = run time.

When one deals only with planning this distinction is unnecessary. In our context we do not need an extra term for planning an action, it suffices to introduce the notion of the execution of an action, see section III.
Even if an action cannot be executed just this fact can be and often is a new piece of knowledge. For actions which provide information the type of the information may be known at planning time (e.g. a telephone number) while the details of the information are only available at run time (i.e. by an execution of the action).

The assumption that we have only partial knowledge about the world applies in particular to the available actions. An action is not only described by its pre- and postconditions (or effects, resp.) but has in addition parameters. The action " make a telephone call" has e.g. the parameter "phone number" and the precondition "phone available". Such an action may be completely unknown or some of its parameters are missing. On an abstract level one may still work with unknown or partially known actions; the execution requires, however, that the action is completely known.

This is one more reason to deal with planning on various levels of abstraction. Abstraction in the context of planning has some tradition, for recent developments see e.g. [Bergmann 96].

It should also be remarked that terms like information and knowledge are used in rather different ways. In [Aamodt 94] e.g. certain relations between data, information and knowledge are investigated. For our purposes it suffices that information is considered as knowledge (in the ordinary sense) about the model. A new piece of information adds something to this knowledge which could not be inferred before. This may be in a twofold manner:
(a) Information about the existence of actions or about the values of its parameters may be added. This enlarges the space of possible actions.
(b) The uncertainty of the preconditions or the outcome of actions is reduced. This reduces space of possible actions.
At planning time one might operate with an incompletely known action as long as a corresponding information providing action which completes the operator description is included before that action. Information providing actions give only new information if they are executed. Because the knowledge base is changed by such actions the knowledge base may be partially contained in the model describing the world.

All the situations discussed above have in common that there is a notion of what a problem and what a solution is. Solutions are taken from a solution space the elements of which are correct solutions, approximate solutions, or non solutions. The solution space may be ordered by a preference relation. The problem space may also be partially ordered expressing that one problem is more difficult than another problem.

All these notions can be modeled in various ways, e.g. deterministic, indeterministic or stochastic. In the following sections we will present simple formal notions for such models. The intention is to provide a basis for discussing the various problems mentioned. Of course, by far not all problems will occur in one application and we will therefore discuss only relatively simple problems types from the viewpoint of an application.

III. Basic formal notions

Our formalisms assume a multiple world model in the sense of modal logic:

$$W = < (W_i)_{i \in I}, \leq >$$

where "$\leq$" is the accessibility relation. In addition, we have actions g, $g \in A$, which provide an index set to $\leq$, i.e. we talk rather about certain $\leq g$ than about $\leq$. Instead of $W_i \leq g\ W_j$ we write

$$W_i \to_g W_j.$$

The worlds W_i may be of very general character containing e.g. real numbers and functions etc. and are hence not restricted to predicate logic.
Actions g are in general described by parameters and may contain variables, i.e. they are of the form $g(p_1,...,p_n,x_1,...,x_m)$. The variables play the usual role as in planning while the parameters are part of the operator description of the action.

We further assume a language $\mathcal{L}$ the symbols of which are interpreted in W. Not all objects (e.g. relations, actions etc.) are requested to have names in L. The intention is that those objects are unknown; at some time they may be introduced into the language. Therefore the language itself is of dynamic character.
The interpretation assigns for the expressions in $\mathcal{L}$ one of the following:

- objects in W
- truth values
- probabilities
- other values depending on the application.

The intention is to keep the model open and flexible for dealing with problems mentioned in the introduction.

Now we introduce some special sets of expressions.

a) Information pieces I; the set of information pieces is the information space II.

b) The knowledge base KB, the expressions of KB are assumed to hold in the world W.

c) Problems P; the set of problems is the problem space IP. Usually the planning problems are (partial) descriptions of a pairs of worlds (W_i, W_j), W_i the initial and W_j is the final world.

d) Possible solutions L; the set of possible solutions is the solution space $\mathbb{L}$.
Usually in planning possible solutions are sequences of actions.
e) Correctness conditions C(P, L) taken from a set $\mathbb{C}$.

We now assume that some information I can be represented in W and that certain expressions in $\mathcal{L}$ can refer to information I. In particular, as indicated above, certain actions may introduce (or rather their effects, see below) other actions). We assume, however, that self references which lead to paradoxes are avoided.

Because the system's solution of problems depends on the available information and in our framework this information may increase during the solution process we have to generalize the concept of a problem.

Def.1: (i) A task is of the form

$$T = (KB, I, P, \mathbb{L}, C(P, L))$$

(ii) $L \in \mathbb{L}$ is a solution of T

$$\leftrightarrow$$

$$KB \cup I \models C(P, L)$$

This means that the correctness conditions assure that possible solutions are really solutions. In principle, there could be solutions which do not satisfy condition (ii) but we did not admit them because their correctness would depend on guessing. This means our solutions are provably correct.
An intermediate set between the correct and the possible solutions is the search space S. The search space is the set that is investigated by the problem solver. Therefore it is not defined in an absolute manner but depends not only on the problem solving method but also on the actual state of the solution process. Hence there are various possible characterizations of it which we not investigate here, however. The only condition which we will assume is

$$\{L \mid KB \cup I \models C(P, L) \} \subseteq S \subseteq \mathbb{L}.$$

The search space gives rise to extend the notion of a task.

Def. 2.: An extended task is of the form $T_E = T = (KB, I, P, \mathbb{L}, C (P, L), S)$ where S is called the search space.

It is often not distinguished between tasks and extended tasks. Next we introduce a new concept for actions, the execution of action. This is done in order to discuss formally the planning and the execution of actions at the same level.

With actions $W_i \rightarrow_g W_j$ sets of expressions are connected as usual:

a) preconditions (denoted by pre(g))

b) postconditions (denoted by post(g))

Pre- and postconditions are extended to sequences of actions. Next we extend the notions for actions:

1) With each action g we associate another action execution(g).
These actions are external and not in the model.

2) execution(g) has

a) preconditions (denoted by pre(g))

b) effects (denoted by eff(g))

3) There is a special precondition of execution(g) denoted by executable(g) which always has to hold.

4) The effects of g is either

(i) a) the transformation of a problem P into a problem P_g;

b) the addition of a new information I_g;

c) the transformation of an extended task

$T = (KB, , P, \mathbb{L}, C(P, L), S)$

to the task

$T_g = (KB, I_g, P_g, \mathbb{L}_g, C(P_g, L), S_g)$

or

(ii) the new information $\neg$ executable(g). In this case we say that the execution of g failed.

5) If executable(g) holds then always 4(i) takes place, otherwise 4(ii) happens.

6) If 4(i) takes place then

task T is solved $\Leftrightarrow$ task T_g is solved.

The corresponding notions for tasks only are obvious. Often it is tacitly assumed that the preconditions imply executability. A discussion of the relations between these notions is given in [Arnold 96]. Actions with $P_g = P$ are called information providing actions. Because we will deal with problems with incorrect information our knowledge base may not only contain expressions that are true in the world under consideration. This implies that e.g. $KB \cup I \models$ executable(g) may hold, without g being executable.

We will have, however, a subset TrueKB of KB which contains only expressions true in the world. For the sequel we assume TrueKB = KB unless specified otherwise.
Costs and benefits of actions will be treated by updating lists which are part of the state of the world.
The most important conditions are 4) and 6). They imply that no backtracking for executed actions is possible. Executed actions and their effects cannot simply be removed, only other executed actions can change the effects. Because executed actions also change the problems we can subsume here that in addition costs for executed actions cannot be removed. This is different from planned actions; if planned actions are removed we can simply forget them. Now we will discuss costs and related aspects more closely.
Technically, in the task description there should be a slot in the knowledge base for the costs (and benefits). This entry is usually initialized with 0 but then updated after each execution of an action. This bookkeeping has no influence on the problem solving itself but it is often required that finally the total costs are stated.

Often, the effects of execution(g) are exactly those which can be inferred from
$KB \cup I \cup$ post(g). There are two important exceptions for this:
a) $KB \cup I$ is unreliable. In this case the effects of execution(g) may even
contradict to post(g)
b) $KB \cup I$ is incomplete.
In this case the effects of execution(g) extend post(g).

Information providing actions g are usually connected with b); the whole purpose of such g is to execute them in order to obtain the desired information. It is important that often some part of eff(g) is contained in post(g). If g is e.g. "look up the phone number of X" then eff(X) is twofold:
a) The fact that the phone number of X is known;
b) the precise phone number of X.
The information contained in a is sufficient to plan further actions following g.

IV. Dependencies

In particular in nonlinear planning various types of dependencies occur. We consider two types.

IV.1 Relations are between actions

Def. 3:

(i) A causal link between g_1 and g_2 denotes that the postcondition of g_1 establishes a precondition of g_2.

Notation: $g_1 \rightarrow_c g_2$.

(ii) g_1 is a thread for g_2 if g_1 destroys some precondition of g_1.

Notation: $g_1 \rightarrow_t g_2$.

In addition, one has the notion that g_3 is a thread for a causal link between g_1 and g_2 of g_3 destroys the precondition of g_2 established by g_1.

We have now two new types of dependencies. One comes from information providing actions and the other results from incomplete knowledge.

Def. 4:

(i) g_2 is information dependent on g_1 if eff(execution(g_1)) provide

(a) some missing precondition of g_2

or

(b) some part of the operator description of g_1 .

Notation: $g_1 \rightarrow_i g_2$

(ii) g_2 is negative information dependent on g_1 if eff(execution(g_1)) yields that some precondition of g_2 fails.

Notation: $g_1 \rightarrow_{ni} g_2$

Due to the incompleteness of knowledge and due to the fact that the precise content of eff(execution(g)) may be unknown each of the notions above has a companion with the add-on “possible”. Hence we have the notions
“possible causal link”, “possible thread” and “possible information dependent”
with the obvious definitions.
Notation: $g_1 \rightarrow_{cp} g_2$, $g_1 \rightarrow_{tp} g_2$, $g_1 \rightarrow_{ip} g_2$.

A special case is when g_2 is itself the execution of some other action g_3. In this case execution (g_1) has executable (g_3) in its effect (resp. ¬ executable (g_3)). It can happen that an ordinary action is a thread to an information providing action, see e.g. example 1 in section IX.

The storage and treatment of causal links and threads is standard in SNLP. For the corresponding notions with respect to information actions as given in definition 4 this is somewhat different. We will first discuss some immediate consequences and then return to the planning task in section VII.

a) The relation $g_1 \rightarrow_i g_2$ gives rise to a knowledge goal.
b) The relation $g_1 \rightarrow_{ni} g_2$ is a thread on an abstract level. Its precise form is unknown and it gives again rise to a knowledge goal if this information is relevant in order to remove the thread.
c) All of the relations give $g_1 \rightarrow_{cp} g_2$, $g_1 \rightarrow_{tp} g_2$, $g_1 \rightarrow_{ip} g_2$ generate candidates for knowledge goals.

Whether it is worthwhile to achieve these various kinds of knowledge goals depends on the actual planning status. In this status the available actions which in principal could be executed are partitioned into four sets:

(I) Actions known as executable (including those where other actions with a causal link have to be carried out first).
(ii) Actions that are possibly executable.
(iii) Actions with a possible thread.
(iv) Actions with a thread.

IV.2 Relations between knowledge and actions

In STRIPS the preconditions are sufficient for the execution of an action. There are, however, conflicts in so far as several actions might be executable in some situation. The selection of a single action is usually based on a strategy or heuristics which looks at some specific characteristics of the situation. These are the "reasons" for the particular choice and are called design rationales.

One way to formalize this is as follows. We assume that the solution space consists of sequences of actions. Suppose T is a task; recall that A is the set of actions.

Def.:The conflict of T is conf(T) = $\{g \in A \mid KB \cup I \models pre(g)\}$.

Def.:

(i) A heuristic is a partial mapping H : $\underline{E} \rightarrow A$, where $\underline{E}$ is the set of sets of expressions.

(ii) H is applicable for T if there is a unique maximal $E \in$ dom(H) such that $E \subseteq KB \cup I$ and $H(E) \in$ conf(T). In this case H(E) is selected as the next action and E is called the *design rational* e for this decision.

The design rationale E should make reference to the conflict and as well as to the preferences applied (see section V.).

The last concept has no counterpart coming from any missing information or uncertainty. After solving the conflict one would now proceed to solve the new task T_g. It should be noted that backtracking is possible as long as g is not executed. The design rationales are important in three main situations:

(a) An inconsistency in the planning occurs and one has to select an intelligent backtracking point.

(b) Due to the execution some external action some part of the knowledge base or the available information is changed and one has to react on this.

(c) One wants to reuse an old plan for a new task.

In (b) the old task T is replaced by the new task T_g. In (c) the reuse is extended to speed up the solution transformation using design rationales (called *complete replay*, cf. [Munoz 96]).

The notion of a design rationale has also an extension to situation with incomplete information: It may happen that the design rationale includes a certain hypothesis on which the heuristics depended. This should be noted in the record of the design rationale.

These situations ask for some truth maintenance system. An advanced such system is REDUX (cf. [Petrie 91]): it has been applied e.g. in the configuration system IDAX (cf. [Paulokat 95]), in the planning system CAPlan (cf. [Weberskirch 95]) or to software and knowledge engineering purposes (cf. [Maurer 94]).
The principal structure of REDUX allows the inclusion of hypotheses about the partially known world. The following changes are appropriate, however:

1) Design rationales should be marked if they include a hypothesis.
2) If all hypotheses are confirmed then the mark is deleted.
3) If a hypothesis is rejected then the same mechanism is applied as in the case where an assumption is withdrawn.

V. Preferences and optimality

In the previous sections we have described the basic notions for actions which change the state of the problem and the state of the information available. It was important to distinguish between actions which are simply planned and those which are executed. These notions have some impact on how optimality issues are addressed.
Optimality is always connected with costs and benefits. Cost functions are in general utilities. Utility functions are, however, often not directly available. What is present is a partial ordering (a preference relation). In our context we encounter several such preference relations.
The domain dependent preferences which describe the costs and benefits of the actions defined for the specific domain under consideration is assumed to given by some partial ordering " ≤ " which we will discuss furthermore here. Instead we consider two relations connected with partial information.

<u>Def.6:</u> The action g is *search space reducing* for the task
T = .(KB, $\mathcal{I}$, P, $\mathbb{L}$, C(P, L), S).if T_g = (KB, I_g, P_g, $\mathbb{L}_g$, C(P, L), S_g) such that $S_g \subset S$ holds.
Notation: $T \leq_{s,g} T_g$.

<u>Def.7:</u> The action g is *solution space expanding* for the task T = (KB, $\mathcal{I}$, P, $\mathbb{L}$, C(P, L), S) if T_g = (KB, I_g, P, $\mathbb{L}_g$, C(P, L), S_g) such that $\mathbb{L} \subset \mathbb{L}_g$ and $S_g \cap \mathbb{L} = S$ hold.
Notation: $T \leq_{e,g} T_g$.

The intention is that g provides information about new actions which enlarge the search space (if at all) only in so far as the new actions are involved.

As in the previous section, these concepts have their counterpart with respect to incomplete information which leads to the concepts (with the obvious definitions):
possibly search space reducing and *possibly solution space expanding,*
Notation: $T \leq_{ps,g} T_g$, and $T \leq_{pe,g} T_g$.

The advantages of search space reducing actions are obvious. Solution space expanding actions may be useful in two situations:
(1) There are no other meaningful actions available.

(2) The available actions will lead to unacceptable bad solutions.

VI. Rules and constraints

It is not our purpose here to discuss rules, constraints and other knowledge representation methods. For the representation of cases it is, however, useful to distinguish between rule and constraint oriented problems. We consider tasks in which a number of variables, say $x_1,...,x_k$ are involved. The values of some of the variables are given (these are called problem variables) and the ultimate task is to determine the value of the remaining variables (these are called solution variables)
Suppose a class $\underline{T}$ of tasks is given.

Def.: The class $\underline{T}$ is called to be of rule type if the set of problem variables is fixed for all $T \in \underline{T}$; otherwise it is of constraint type.

In a more general setting the problem variables of a problem variables of a problem are not completely specified but only restricted by some additional constraints. Another important generalization is when constraint problems contain optimization aspects. One way to handle this is to distinguish between hard and weak constraints and order the latter ones by priorities, as done in the CONTAX-system, cf. [Meyer 95].

VII. Planning and knowledge goal planning

Here we will draw some first consequences of the framework introduced so far to integrate the achievement of knowledge goals in an SNLP-planer. It should be observed that knowledge goals have no value in themselves but are only means to achieve other goals.
A central part of an SNLP-planer is the algorithm which arranges the ordering of actions in such a way that threads disappear. We need to distinguish two aspects:
(a) Establishment of correctness conditions with respect to the executability of actions.
(b) Selection of actions with respect to preference relations.
In both aspects knowledge goals occur explicitly and have to be achieved.
In aspect (a) the dependency relations of section IV.1 play a role. The recording of these relations is the same as in ordinary SLNP-planning in so far as just some

abstract dependencies have to be stored. Also the top-level algorithm of SNLP can be taken over.
On the next level of detail the partition of space of available actions mentioned in IV.1 play a role. This first influences the ordering in which the algorithm consideres the actions; this will just lead to specific heuristics. Next the partition gives rise to activate information providing actions:

(i) Information dependencies give rise to knowledge goals as subgoals.
(ii) Negative dependencies give rise to knowledge goals only if its information is needed on the top level.
(iii) "Possible" relations can be used either for the reduction of the search space by assuming that causal links are not established or threads are already there or for the generation of knowledge goals.

In which way this is realized in detail again depends on special heuristics.

For aspect (b) the notions from section IV.2 are relevant. The search for better solutions leads to an investigation of the design rationales an in particular of the possible alternatives. The wish to select better alternatives leads again to new knowledge goals.

If the achievement of knowledge goals is a major task and does not only result in a simple query then the planning has several specific characteristics. One is that the information is stored at different locations which have specific access paths.
A system which realizes information retrieval in a general sense is COBRA, see [Carranza 96].

VIII. Cases

Traditionally in case-based reasoning cases are ordered pairs of the form (problem, solution). This was often certainly adequate. Examples are classification tasks or certain actions for a decision maker had to be generated. In planning one often considers cases of the form (problem, partial solution). This is of interest if one wants to partially complete partial solutions. The situation is even more different if the task is not of rule type or the effects of some actions in the solution are not deterministic. Also we deal with incomplete information in planning. For simplicity we assume an attribute-value representation.

VIII.1 Incomplete information

Cases have been used in CBR-systems for diagnosis for the partial completion of information, cf. [Wess 94]. There are basically two ways a CBR-system can handle incomplete information:

(1) Similarity measures accept input tuples with unknown values.

(2) Cases can be used to select the next variable for which a value has to found.

(1) The handling of unknown values in measures is a very delicate matter. In the measure used in INRECA (cf. [Althoff 95]) it leads to an unsymmetry of the measure because the impact of a missing value in a query case is not the same as a missing value in a case from the case base. This is also connected with some implicit knowledge about the expectation of the missing value itself as well as about its importance. In principle, unknown values could give rise to knowledge goals. The treatment in similarity measures avoids this because sufficient a priori knowledge is compiled into the measure.

The handling of unknown values in similarity measures for planning seems to be not different from the diagnostic situation.

(2) Those cases which are used for selecting the next variable to be associated with a value are called strategic cases because in diagnostics they determine the diagnostic process.

In the usage of strategic cases we have to distinguish the aspects mentioned in section VII; if only “good” cases are stored this is of minor importance.

The selection of needed information can be coupled with the replay technique. In a replay the plan of a case from the case base is applied as long as possible to an actual problem; if this is no longer possible then generative planning takes place (as in the CAPlan-system, cf. [Munoz 96]).

Here the replay may stop due to missing information. It can first be continued with preferences obtained from the partitioning of the action space described in section IV.1 as if those action that are not completely ruled out where still there. In a second step this leads to knowledge goals in order to confirm the missing information.

Finally we remark that in order to reuse cases for information retrieval it is often more advisable not to include the detailed information itself in the case but rather the location where the information is stored.

VIII.2 Constraint type cases

Constraints typically occur in design, e.g. in the design of buildings. A superficial view could suggest that a case is the tuple of all values for the variables because after all this tuple describes the result the customer wants to have. If CB contains only correct solutions then this would suffice as long as only correctness is involved. When optimality is also required then this becomes insufficient even if CB contains only cases with optimal solutions. The reason is that optimality is defined with respect to the values or constraints of the problem variables. This leads to the following definition.

Def.: A case of constraint type is a marked tuple of values for all variables. The marks select the problem variables and state their constraints in the original problem.

In many applications the problem variables are quite arbitrary but more or less determined by a user type. Each user type has a different view on the cases. If the objects are e.g. houses then we have among the user types architects, builders, customers who by the house and banks which do the financing. Each group selects certain variables as problem variables, has its own preferences and its own default values. This observation allows to proceed in a somewhat restricted way.
With each user type ut we associate the following.
Notation:
(i) V(ut) is a tuple $(x_1,...,x_k)$ of variables which are called the problem variables of ut.
(ii) sim_V is a similarity measure on the tuples of variables.
(iii) sim_{ut} is a similarity measure on the on the k-tuples of values from the domains of the variables x_i from V(ut).
(iv) Each k-tuple of values from the domains of the variables x_k has either the label “default” or “exceptional”.

Comments:
(i) V(ut) is intended to contain those variables which users of this type are most likely to select as problem variables.
(ii) This measure is applied when the type of some user is not completely clear. The type of an actual user is determined using the nearest neighbor method with respect to sim_V. In addition even users of the same type do not choose

always the same problem variables. This approach can also be regarded as some kind of preprocessing for case retrieval.

(iii) Each such measure reflects the view of its user type. Such individual measures for user types are of course applied very often. The specialty here is that these measures are applied on different argument sets.

(iv) This should indicate whether the user wishes are considered to be normal or unusual. The intention is not that exceptional values are considered as “second class citizens”. The idea is rather that one should not use such values for actual problems with care because they reflect unusual demands. If e.g. the case describes a building with a variable for “window type” then an exceptional value would be a very expensive one. The label reminds to the fact that this value was not selected during the solution process but given by the customer.

It should be noted that a more adequate would be to model V(ut) as well “default” and “exceptional” as fuzzy sets. The approach taken here simulates the fuzzy membership functions by CBR-techniques.

VIII.3 Cases, effects and multiple cases

The intended problem is that of a decision maker who has to produce an action as the solution for the task. The case description of the form (task, action) is insufficient if the outcome of the action is uncertain, in particular if the action may not be executable. In [Gilboa 92] it was suggested to represent cases as triples; due to the lack of a common term we call them extended cases.

Def. An extended case is of the form (task T, action g, eff(g)).

A first observation is that there may be two cases coinciding in the first two entries but with different effects; without the effects these two cases could not be distinguished.

This leads to question whether one would admit the same case twice in the case base. In a deterministic situation with complete information there would be no need for this. In a more general context the occurrence of several cases with identical description may, however, be highly informative.

Def.: A multiple case is a pair (C, n) with C a case and $n \geq 1$.

The information in a multiple case is that values unknown at the time of problem solving occurred n times. There are two major situations where this is important:

(i) Some value important for the solution is missing and one still has to proceed (e.g. to apply a suitable information providing action).
(ii) A decision has to be made in form of some action with an uncertain outcome.

In both situations the multiple occurrence of cases will increase the likelihood of certain values and will therefore give rise to certain preferences of actions. As long as the occurrences are just recorded in the case base their information is used at run time. In the terminology of [Richter 95] the information is stored in the container "case-base". If sufficiently much of such information has been obtained it may be suitable to shift it to the container "similarity measure" which means that it is handled at compile time.

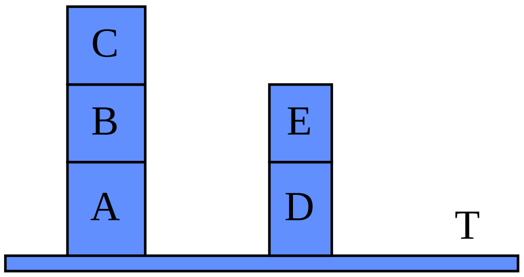
IX. Examples

We will now give simple examples from the blocks world in order to illustrate the interplay between ordinary planning, information providing actions and abstraction in the presence of costs.
We use A,B, C,.. for blocks, X,Y,Z,... as variables for blocks, T is the table. We have the predicates ON(X,Y) resp. ON(X,T) and CLEAR(X).
Actions are of the form PUT(A,B) and PUT(A,T). Information providing actions are queries of the form ON(A,B)? For both types of actions no variables are allowed.
PUT-actions are executable if the usual preconditions are satisfied (what may not be known); queries are always executable.
Besides the usual axioms for the blocks world we assume that towers are of height at most 3.
The cost structure is given by:
cost(query) = 1 unit;
cost(other action) = 4 units;
cost(false solution) = 20 units.
Costs will apply of course also to failed executions of actions.

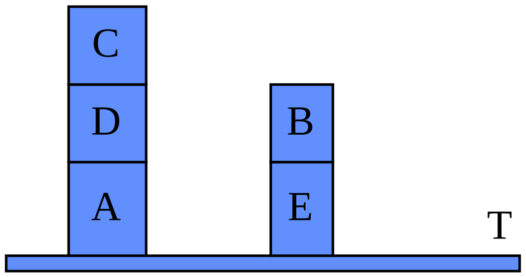
1. Example.

The numbers in the sequel refer to the comments at the end of this section.

Initial situation:
ON(C ,B), ON(B,A), ON(A,T), CLEAR(C), ON(E,D), ON(D,T), CLEAR(E).
Goal situation:
ON(C,D), ON(D,A), ON(A,T), , CLEAR(C), ON(B,E), ON(E,T), CLEAR(B).

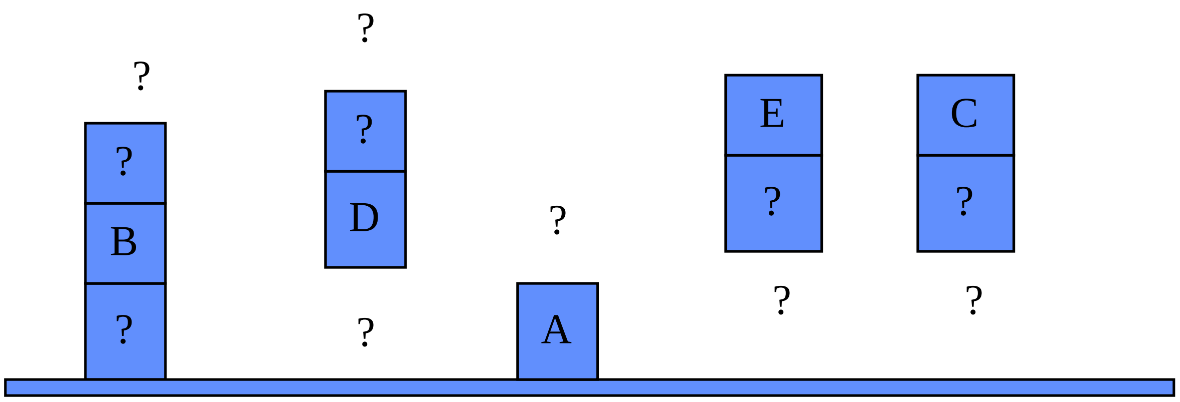


Initial Situation



Goal Situation

We observe that some information is redundant. Next we introduce an incomplete information about the initial situation; we assume that this information is reliable, all blocks are different and no unknown blocks exist.
Given information about the initial situation:
ON(X,B), ON(B,Y), ON(Z,D), ON(A,T), ON(E,U), ON(C,V), CLEAR(E), CLEAR(C).
The variables are here understood as existentially quantified.



A first analysis of the situation yields the following:
X ∈ {E,C}, Y ∈ {A,D}, Z ∈ {B,E,C}, U ∈ {B,D,A}, V ∈ {B,D,A}, (see (1)).
There is a choice now between a PUT-action and a query.
a) PUT-actions: The only one which can be surely executed is PUT(E,T). This action has necessarily to be executed at some time. It prevents us, however, to rise queries

of the form ON(E,.)? Therefore it may make impossible to find out what U is (unless U = X) or what X and Z are (because E = X or E =Z may hold) or what might be on top of A, (2). All other PUT actions will fail with a high probability (unless information on the probability distribution of the situations is given).
b) Queries: On verifies that at least two queries are needed. It suffices to rise the queries ON(C,B)? and ON(B,A)? in order to get complete information. The ordering of the queries is irrelevant although both ways lead to different considerations, (3).
Although ones knows that there is a complete information achieved there are too many possibilities to continue planning on the concrete level. On an abstract level the following plan is possible:
(i) ON(C,B)?
(ii) ON(B,A)?
(iii) Solve the remaining task.
In order to make more detailed planning one needs to execute the first two steps:
(i) execute(ON(C,B)?)
(ii) execute(ON(B,A)?).
This changes the task and one can proceed as usual (4).

2. Example.
We vary the first example such that each PUT-action shows what was underneath (either the table or the block) with no extra cost.
This implies that PUT(E,T) provides complete information and is therefore the optimal action. To continue planning again execution(PUT(E,T)) has to be done (5).

3. Example.
This example shall only demonstrate that even in the blocksworld there may be a need for planning information retrieval carefully. We will only state a problem situation without going into the details of the solution.
We now assume that the goal situation is given but nothing is known about the initial situation. Besides the actions in example 1 we assume some more actions:
TOP? : lists all top blocks.
NEXT? lists all blocks directly under top blocks.
SHOWBLOCK(A): gives the blocks below A as a set, if A is some top block, otherwise it fails (for each block A).
Cost structure:
Cost(TOP?) = 5;
Cost(NEXT?) = 5 if directly asked after a TOP?-query, otherwise 8.

Cost(SHOWBLOCK(A)) = 5.
(See (6)).

Comments:

(1) This is achieved by constraint propagation.

(2) This means we have a possible thread between PUT(E,T) and queries of the form ON(E,.), ON(.,B), ON(.,D) and ON(.,A).

(3) This is the (optimal) constraint satisfaction problem which occurs in diagnostic processes: What is the optimal information needed?

(4) This illustrates the interleaving between planning and executing actions in the context of information retrieval.

(5) PUT(E,T) is an “ordinary”action as well as a information providing one.

(6) This example illustrates that information retrieval has to plan the access to the different locations where the information is stored.

Bibliography

[Aamodt 94] A.Aamodt, E.Plaza: Case-Based Reasoning: Foundation issues, methodological variations and system approaches. AI-Communications 7 (1994).

[Althoff 95] K.D.Althoff, E.Auriol, R.Bergmann, S.Brean, S.Dittrich, R.Johnstone, M.Manago, R.Traphoener, S.Wess: Case-Based Reasoning for decision Support and Diagnostic Problem Solving: The INRECA Approach. Proc. 3rd German Workshop 0n CBR, 1995.

[Arnold 96] O.Arnold: Die Therapiesteuerungskomponente einer Wissensbasierten Systemarchitektur fuer Aufgaben der Prozessfuehrung. DISKI 130, infix-Verlag 1996.

[Bergmann 96] R.Bergmann: Effizientes Problemloesen durch flexible Wiederverwendung von Faellen auf verschiedenen Abstraktionsebenen. DISKI 138, infix-Verlag 1996.

[Carranza 96] C.Carranza, W.Lenski: A Planning-Based Approach to Intelligent Information Retrieval in Text Databases. Proc. 20th Annual Conference Ges. f. Klassifikation, Springer-Verlag 1996.

[Gilboa 92] I.Gilboa, D.Schmeidler: Case-Based Decision Theory.

[Maurer 94]: F.Maurer, J. Paulokat: Operationalizing Conceptual Models Based on a Model of Dependencies. Proc. ECAI 1994 (11th European Conference on Artif. Intell.)

[McAllester 91] D.McAllester, D.Rosenblitt: Systematic Nonlinear Planning. Proc. AAAI-91.

[Meyer 95] M.Meyer: Finite Domain Constraints: Declarativity meets Efficiency - Theory meets Application. DISKI 79, infix-Verlag 1995.

[Munoz 96] H.Munoz-Avila, F.Weberskirch: Planning for Manufacturing Workpieces by Storing, Indexing and Replaying Planning Decisions. Proc. Third International Conference on AI Planning Systems, 1996.

[Paulokat 95] J.Paulokat, F.Maurer: Entscheidungsorientierte Rechtfertigungsverwaltung zur Unterstuetzung des Konfigurationsprozesses in IDAX. In: Expertensysteme 95 (ed. M.M.Richter, F.Maurer), infix-Verlag 1995.

[Pryor 95] L.Pryor: Decisions, decisions: Knowledge goals in planning. Hybrid Problems, Hybrid Solutions (ed. J. Hallam et al.), IOS Press 1995.

[Richter 95] M.M.Richter: Invited talk at ICCBR’95, Sesimbra, Portugal 1995.

[Weberskirch 95] F.Weberskirch: Combining SNLP-like Planning and Dependency-Maintenance. LSA-Report LSA-95-10E, Univ. Kaiserslautern 1995.

[Wess 995] S.Wess: Fallbasiertes Problemloesen in wissensbasierten Systemen zur Entscheidungsunterstuetzung und Diagnostik. DISKI , infix-Verlag 1995.