Planning With Case-Based Structures

P.V.S.R.Bhanu Prasad

Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Madras - 600036
India
bhanu@iitm.ernet.in

Abstract

We present a case-based planning (CBP) model. The important component in the memory of the system is the collection of different *structures* of created plans. A structure is made up of created plans arranged in a hierarchical fashion, and corresponds to a class of plans. Retrieving, modification, and storing mechanisms play a key role in plan construction. In contrast with most existing CBP systems, these mechanisms are distributed over plan operators. A plan is created in a hierarchical fashion by activating the suitable structure and performing stepwise retrieving, modification, and storing operations on it. The system learns new plans from experience and has been implemented in the domain of cooking vegetables.

Introduction

The focus in classical planning has been on generating plans from scratch by searching the space of partial plans (Tate 1975, Sacerdoti 1987, Wilkins 1988). Recently due to the developments being made in the areas of cognitive science and machine learning, designing planning systems that use case-based approaches has become an attractive option. Many systems have been built (Kolodner 1987, Hammond 1989, Redmond 1990, Subbarao 1992, Kettler et al. 1994) within the framework of memory organization packets(MOPS) (Schank 1982, Riesbeck 1989). In all these systems, entire plans are stored as cases to be retrieved *in toto*. A new plan is created by adapting a suitable plan or *pieces* of some plans merged together from memory to achieve the given goal. The issues of storing newly created plans in memory and the retrieval of plans from memory are the key factors for these systems. Learning is an other important characteristic for a CBP system.

Present CBP systems (Kolodner 1987, Hammond 1989, Redmond 1990, Subbarao 1992, Kettler et al. 1994) have treated each of the following issues independently and less emphasis has been given in developing a unified system.

(i). *Cost of plan modification* (Hammond 1989): Adapting a plan or merging the pieces that needs less modification cost.

(ii). Hierarchical planning (Sacerdoti 1974, Tate 1975, Stefik 1981, Sacerdoti 1987, Wilkins 1988):
To solve hard problems, a problem solver may have to generate long plans. In order to do that efficiently, it is important to be able to eliminate some of the details of the problem until a solution that addresses the main issues is found (Rich 1983).

(iii). Knowledge sharing:
Optimum sharing of subgoals is an important characteristic for an efficient knowledge-based planning system.

In this paper we explore the design of a unified system that addresses the above issues.

The memory of our system is organized mainly using the created plans arranged in a hierarchical fashion. Domain object specific tailoring is done with the aid of a memory of the properties of domain objects organized in an inheritance hierarchy. We treat the terms abstraction hierarchy and inheritance hierarchy synonymously.

The system has been implemented in the domain of cooking vegetables in the Indian style. The motivation for taking up the culinary domain stems mainly from its wide access and its similarity to more complex domains, such as circuit construction (Hammond 1989). The cooking domain is well represented in the case-based reasoning community (Kolodner 1987, Hammond 1989). The following sections illustrate the memory organization and the process of plan creation.

Memory Organization

The underlying assumption is that there exists distinct and identifiable styles of cooking. A style can be accessed using the root node of a structure, and the root node is reached by a simple indexing mechanism. A plan is created by expanding the corresponding structure which is guided by knowledge of ingredients used to tailor the actions for the given ingredients. Two separate memories, namely the memory of properties and the memory of plan structures involve in plan generation.

Memory of Properties

The memory of properties is organized using an abstraction hierarchy. At the top of the hierarchy are more generalized properties. These properties inherited by all the corresponding specializations. The properties are used to index modification rules to tune the plans (*e.g.* recipes) to specific domain objects (*e.g.* vegetables). The names of the domain objects occupy the bottom most positions in the hierarchy. Thus in this model, a domain object is represented as a specific collection of properties needed for tuning the plans.

Memory of Plan Structures

The planning knowledge is organized based on the fact that there are different recipe *styles* which are generally different from each other. At the same time, within the same style different vegetables can be cooked in a similar way. For example, the recipe for brinjal fry is very different from brinjal chutney, but quite close to potato fry. The memory of structures captures all the created plans. Each structure corresponds to a class of plans and is made up of hierarchically organized created plans.

A structure is identified with the highest-level plan operator such as fry. This operator in turn has indexing links (Riesbeck 1989) to the set of plans that are created in terms of the operators at the next hierarchical level. For example, the operator *fry* has indexing links to the following plans:
(i) *basic-preparation, prepare-pieces, preparation-for-fry, perform-fry* (corresponds to lady's finger).
(ii) *basic-preparation, boiling, preparation-for-fry, perform-fry* (corresponds to peas).

The properties of the domain objects that are involved in these plans are the indices for these links. We call each plan in this set as a *next-level* plan of the operator. Again each operator in these next-level plans may have a similar type of indexing links. This arrangement is continued until the primitive level operators are reached, which are the executable level (e-level) operators. The properties that distinguish the next-level plans may be different for different operators.

The properties corresponding to the operators at lower hierarchical levels are relatively unimportant compared to those at the higher-levels, and the consideration of the former is postponed until most of the plan is generated. In other words, plan generation occurs in stages, dealing with the highest-level properties first. Once a plan has been generated by considering these highest-level properties, other levels will be generated by considering the properties at subsequent levels.

The plans share their common subgoals. In the above example, the operators namely *basic-preparation*, *preparation-for-fry*, and *perform-fry* are common to both the lady's finger and peas. The terms plan operators, and subgoals are used interchangeably in this paper. The e-level operators have no indexing links.

The system starts with some plans that are supplied directly by the creator. In contrast with CBP systems like CHEF (Hammond 1989), some (or even all) of the initial plans may be incomplete. In other words, the next-level plans of different subgoals may correspond correspond to different vegetables. The only restriction is that each operator is provided with at least one next-level plan. For example, while starting with this system there is only one next-level plan for fry that corresponds to lady's finger and there is only one next-level plan for boiling that corresponds to peas.

Retrieving, modification, and storing mechanisms play a key role in plan construction. In our system all these mechanisms are distributed over plan operators. In other words, each operator has its own retrieving, modification, and storing mechanisms. Now we see each of these mechanisms.

Retrieving Mechanism

The retrieving mechanism of a subgoal identifies and retrieves a suitable next-level plan for the subgoal. In our system, the properties (only those distinguish the next-level plans of the subgoal) are arranged in the form of a discrimination net associated with the subgoal. Similar to CHEF (Hammond 1989), the properties are ordered on the basis of their modification cost. The higher-priority ones are used at the higher-levels of the discrimination net. For a given goal, the retrieving mechanism always searches the corresponding discrimination net for a next-level plan that optimally satisfies, important properties of the goal. Let us call

this next-level plan as the *expansion* of the subgoal. Since the retrieving mechanisms are distributed over the plan operators, appropriate (with respect to cost of modification) plan pieces are retrieved at each stage of adaptation.

Modification Knowledge

The modification knowledge associated with a high-level operator performs modifications on the *expansion* to tune it to specific domain objects by incorporating additional operators and/or deleting some existing operators. The modification knowledge of an operator is in the form of a set of *if-then-else* rules. For example, a modification rule associated with fry is: If the vegetable is of *large-size* then add *prepare-pieces* immediately after *basic-preparation*, unless this operator has already appeared in the expansion.

Most of the modification rules are indexed by the properties of the domain objects. As a consequence, a modification rule indexed by a a set of properties is inherited by all other properties and domain objects that are its specializations in the memory of properties.

All modification rules are of *high-level* type. An operator incorporated into a plan by a modification rule will again split down into its expansion, and in turn, each operator in this plan may be broken down similarly and so on. Therefore, it is possible to add or delete an entire set of e-level operators, all at once, simply by adding or deleting the corresponding high-level operator.

Since the modification rules are distributed over plan operators, if an operator is not participating in the construction of a plan then all the modification rules that are associated with that operator and with all the next-level plans of that operator will automatically be omitted from consideration. This will save time while checking for a suitable modification rule for application.

Finally, the modification rules of e-level operators are used to assign values to variables in the e-level plan. For example, quantity of an ingredient, and size of pieces.

Storing Mechanism

The storing mechanism associated with an operator stores a newly created next-level plan at an appropriate location in the discrimination net associated with the operator. The features that are used to store a plan are the same as those used to access it.

The root node of a structure is used as its index in the set of structures available. Since each structure embodies a high-level goal that captures a class of plans, we expect the number of structures will be much smaller than the number of possible plans. Thus we expect a simple indexing mechanism to work, specially since each structure is going to be quite distinct. Once the required structure is retrieved, a plan is created by stepwise retrieving, modification, and storing processes. The structures share their common subgoals. Now we see how a plan is created.

Plan Creation

Plans are created in a hierarchical fashion. Initially, the highest-level operator of a suitable structure is expanded. This plan is modified if necessary. The modified plan is stored appropriately as one of the next-level plans of the operator. A similar process is carried out with each of the operators in this next-level plan. This process is repeated until the e-level plan is generated.

The Algorithm

1). /* INITIALIZATION */

Input ← goal specification. Correspond to each level *i* in the hierarchy, create a list *operator-list-i* to store the plan at that level. For each *i*, set operator-list-i ← NULL.

2). /* STRUCTURE SELECTION */

Select the operator representing the root node of the appropriate plan structure. Add it to *operator-list-0*. Set i ← 0.

3). /* EXPANSION, MODIFICATION, AND UPDATION */

Starting from the first element, expand each element in *operator-list-i*. Apply the relevant modifications. Store the modified next-level plan. Also append the plan operators at the end of *operator-list-(i+1)*.

4). /* DESCENT */

Set i ← i + 1. If all the operators in *operator-list-i* are not e-level then go to step 3. Else output the e-level plan.

The user may not always need plans for achieving *main* goals. The above algorithm can as well be used with slight modifications in order to generate a plan for a subgoal. In this situation it is sufficient to start with the required subgoal rather than the root node in step 2 of the algorithm.

Conclusion

Most of the CBP systems like CHEF (Hammond 1989) have treated each case as a separate entity and per-

formed planning by selecting one. MEDIATOR (Simpson 1985) is one of the earliest systems that utilized pieces in forming a case. However, MEDIATOR had to choose a case in order to access the relevant part.

Even though Barletta and Mark (Barletta 1988) dealt with the pieces, their way of organizing the pieces is different from our approach. In their approach, cases are grouped into pieces in terms of the hypothesized faults. This organization does not allow easy direct access to the pieces of a stored case.

There is a similarity between our approach and CELIA (Redmond 1990): In both the systems case pieces are organized in a hierarchical fashion and these pieces can be accessed either direcly or sequentially. The basic difference is that in CELIA, a link represents a relationship between goals in the case where as in our system a link connects a subgoal and one of its next-level plans.

Though both are hierarchical, our approach of retrieving pieces is totally different from PRIAR (Subbarao 1992). PRIAR essentially retrieves a complete plan from memory for adaptation.

In the present form, our system does not learn from experience. We intend to incorporate a selective learning element which will perform *expectation learning* (Hammond 1989) and *critic learning* (Hammond 1989) by getting feedback from the world.

References

Barletta, R.; Mark. W. 1988. Breaking Cases Into Pieces. In *Proceedings of Case-Based Reasoning Workshop*, St. Paul, MN, USA.

Hammond, K.J. 1989. *Case-Based Planning: Viewing planning as a memory task.* Academic Press, Inc, New York.

Kettler, B.P. et al. 1994. Massively Parallel Support for Case-Based Planning. *IEEE EXPERT, February.*

Kolodner, J. 1987. Capitalizing on Failure Through Case-Based Inference. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum Associates, Publishers, New Jersy.

Redmond, M. 1990. Distributed Cases for Case-Based Reasoning; Facilitating Use of Multiple Cases. In *Proceedings of AAAI.*

Rich, E. 1983. *Artificial Intelligence.* McGraw-Hill Book Company, Singapore.

Riesbeck, C.K.; and Schank, R.C. 1989. *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, Publishers, New Jersy.

Sacerdoti, E.D. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, Volume 5.

Sacerdoti, E.D. 1987. *A Structure for Plans and Behavior.* Amsterdam: Elsevier-North Holland.

Schank, R.C. 1982. *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge University press.

Simpson, R.L.Jr. 1985. A Computer Model of Case-Based Reasoning in Problem Solving. Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, USA.

Subbarao, K.; and Hendler, J.A. 1992. A Validation-Structure Based Theory of Plan Modification and Reuse. *Artificial Intelligence*, Volume 55.

Tate, A. 1975. Project Planning Using a Hierarchic Non-linear Planner, Research Report 25, Department of Artificial Intelligence, University of Edinburgh, U.K.

Wilkins, D.E. 1988. *Practical Planning-Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, San Mateo, California.

Appendix: Guidelines for describing the contribution

A. Reasoning framework

1. What is the reasoning framework?
The framework is case-based reasoning
2. What benefits can be gained that motivated using adaptation? reuse?
It is aimed to reduce the cost of plan modification and flexible utilization of subplans.
3. What are the specific benefits and limitations of your approach?
The approach is suitable for *structured* domains in which domain objects can be represented as having specific collection of properties.
4. What invariants does it guarantee, if any? That is, given representative inputs, what is known to be true about the outputs?
The output is a plan that satisfies the input goal.
5. What are the roles of adaptation, knowledge, and reuse in your approach? We include guidelines for this question below.

B. Task

1. What is the task? The domain?
The task is planning and the domain is culinary.
2. What are the inputs?

The inputs are the description of dishs.
3. What are the outputs?
The outputs are the recipes satisfying the input goals.
4. What constraints are on the outputs (i.e., specifications that must be satisfied)?
The ingredients that should be present in the dish, and their taste etc.
5. Are there characteristics of the domain that the method requires, relies on, or exploits in some way?
This method assumes that each domain object has some specific collection of properties.

C. Evaluation

1. What hypotheses were explored?
The memory organization, storage and retrieval are explored.
2. What type of evaluation?
The evaluation is of psychological type.
3. What comparisons were made with other methods?
Comparisons are made regarding the storage and retrieval of plan pieces, hierarchical organization, and plan modification.
4. How does the evaluation validate or illuminate your theory of adaptation of knowledge for reuse?
The adaptation algorithm has been validated by culinary experts for a number of cases belonging to various styles of dishs.
5. What are the primary contributions of your research?
The work is aimed at developing a new case-based planning system.

Guidelines for Answering Question A.5

A. Knowledge

1. What does it encode?
It encodes the Knowledge of ingredients and plan structures.
2. How is it represented?
The knowledge of ingredients is represented using an abstraction hierarchy. Created plans are stored in hierarchical structures, using indexing links. The modification knowledge is associated with plan operators.
3. How is it used?
By retrieving suitable subplans at each stage of adaptation.
4. How is it acquired?
The knowledge of ingredients and the modification knowledge are hand-coded. Initially some plans are also hand-coded. Other plans will be learned from experience.

B. Adaptation

1. What is adapted?
Subplans are adapted.
2. Why is it adapted?
It is adapted in order to generate new plans.
3. What properties of the knowledge representation does adaptation require or exploit?
The properties of the ingredients involved in the subplans are used as their indices for adaptation. Subplans are checked locally in order to find a closest match for adaptation.
4. How are the benefits measured?
The benefits are not yet measured.
5. What is gained?
Cost effective adaptation.

C. Reuse

1. What is reused?
Earlier subplans are reused.
2. Why is it reused?
In order to generate new plans .
3. What properties does the reuse process place on (the knowledge representations produced by) the adaptation process?
The subplans are reused. The properties of the ingredients involved in the subplans are used as their indices for adaptation. Subplans are checked locally in order to find a closest match for adaptation.
4. How are the benefits measured?
The benefits are not yet measured.
5. What is gained?
Flexible utilization of subplans.