DoLittle: a multi-strategy planning system

Jacky Baltes

April 28, 1997

Keywords: Multi-strategy planning, macro-operators, abstraction, case-based planning.

ABSTRACT

This paper describes multi-strategy planning and its implementation in the DoLittle *system.* Planning strategies *are defined as methods to reduce the search space by exploiting some assumptions (so-called* planning biases*) about the problem domain. General operators are a generalization of standard* Strips *operators that conveniently represent many different planning strategies. The search control method applies different general operators based on a strongest first principle; planning biases that are expected to lead to small search spaces are tried first. An empirical evaluation in three domains showed that multi-strategy planning performed well in all and significantly outperformed the best single strategy planners in some domains.*

1 INTRODUCTION

The classical planning paradigm has long been an active research area in artificial intelligence. The problem is to find a sequence of actions that transforms the world from an initial state into a goal state, given a set of actions that an agent can perform in the world.

One reason for this popularity is that many practical problems can be interpreted as strategical planning problems, e.g., scheduling of machines in a factory, file system maintenance in an operating system, or cargo delivery.

This work aims at developing a strategical planning system for a kitchen robot. To simulate this environment, I developed a kitchen domain; it consists of an one-armed, mobile robot whose task it is to prepare different beverages. The kitchen domain is a complex domain. It contains 51 operators and 45 objects and has an average branching factor of 3.5. Plans contain many primitive operators, e.g., making a cup of tea, which is one of the simplest tasks, takes 30 steps.

Unfortunately, theoretical results show that planning is intractable even in simple domains. To be practical, a planner must therefore reduce the size of the search space. These reductions bias the search by either removing or preferring parts of the search space or restructuring it. Based on the notion of an inductive bias in machine learning, I introduce *planning bias* to describe these assumptions. Examples of planning biases include assumptions about the structure of the search space, the domain description, the plan structure, the problem set, and the order of the problems.

A *planning strategy* is any method that exploits some planning bias. The following paragraphs give a very brief introduction to some popular planning strategies.

Means-ends analysis attempts to solve problems by looking at the differences between the current state and the goal state [2]. A difference is then selected and an operator is added to the plan to reduce this difference. Means-ends analysis works well in domains, where there are few operators to reduce a difference and the order in which the differences are reduced is not important.

Macro-based planning attempts to reduce the length of the solution by reasoning about sequences of operators instead of just primitive operators. It is based on the assumption that a small number of fixed sequences of primitive operators occur very frequently in solutions. For example, when moving an object from one room to another with a closed door, a robot has to execute the following subsequence: turn, put down an object, turn, open the door, turn, pick up the object, and turn to walk through the door. By adding an operator to encapsulate this subsequence, the reasoning process may be sped up through the reduction of the solution length. Macros must balance this reduction with the increase in the branching factor and the matchcost.

Abstraction-based planning first ignores low level details of the problem, to create an abstract solution. The abstract solution is then refined by adding more and more detail until all abstract operators are replaced by primitive ones. For example, assume that the planner is given the problem of moving a robot from Auckland to Banff. An abstract planner would first create a plan with three abstract operators: (a) go from the current location to the Auckland airport, (b) fly from Auckland to Calgary, and (c) go from Calgary to Banff. A planning system that reasons about the primitive actions, such as go forward, turn right, turn left will quickly be overcome by the tyranny of detail.

Case-based planning reuses previous solutions when solving new problems. Given a problem, a similar plan is retrieved and adapted to the new situation.

The adaption methods are specific plan debugging routines. The assumption is that a similar plan can be found efficiently and that the cost of adapting an old solution is less than that of creating one from scratch. For example, in software design people often use somebody else's code and adapt it to their requirements.

Motivation Planning systems based on particular planning strategies work well if the underlying assumptions are met, but fail (often spectacularly) if they are not. Many different planning strategies have been developed. However, no single bias has been found to be superior or even sufficient in all domains. This led researchers to conclude that planners must use different planning biases in different domains [5].

The motivation for multi-strategy planning is that instead of developing a new planning strategy, the planner is based on partially successful, well known planning strategies. So far the work focused on macro-based, case-based, and abstraction-based planning. The problem is to determine when a given planning strategy is appropriate for a domain and solve the problem with it. Unfortunately, there are many examples in which a planning strategy is not sufficient for a domain or even a single problem. Instead, some parts of the problem can be solved efficiently by a planning strategy, but another planning strategy is needed for the remainder. Therefore, a multi-strategy planning system must (a) break the problem up into subproblems, (b) select planning strategies and solve the subproblems, and (c) combine the solutions to the subproblems.

The remainder of this section is a brief example, to give the reader a feeling for the essence of multi-strategy planning. Therefore, the comparison is based solely on the necessary search depth and ignores other factors such as the branching factor. A completely worked example can be found in [1].

Assume that in the kitchen domain, the goal is to prepare a cup of instant coffee with sugar. The solution to this problem contains 42 primitive operators.

Macro-based planning exploits often used operator sequences in the domain. Because of the large variety of possible location for the utensils in the kitchen domain, there are few long recurring operator sequences. The average length of the useful macros in the kitchen domain is about four operators. For example, the following macro fills a cup with water (put cup in sink, fill cup with water, turn water off, pick up cup). Given that macros only contain small number of operators, they can not reduce the search space sufficiently. For example, in this case, the search depth is still at least ten operators.

Abstraction-based planning creates an abstract plan to fill a cup with water: (heat the water by either using the microwave or the stove, add instant coffee, get the sugar jar, add sugar). However, the refinement of each of those abstract operators is non-trivial in itself, and abstraction-based planning does not provide any guidance when searching for the refinement. For example, the refinement of the "heat-the-water" operator consists of ten operators.

Case-based planning retrieves a plan (in this example, the plan for making tea is retrieved) and adapts it. This requires replacing the tea bag with instant coffee. However, a case-based planner has to create a suffix plan to get the sugar jar and open it, and then scoop the sugar into the coffee. This takes an additional 19 steps in the kitchen domain.

Multi-strategy planning uses all planning strategies to make instant coffee with sugar. It uses case-based planning to find an initial plan. However, it is able to employ other planning strategies when searching for the suffix plan to add sugar. An abstract plan to add the sugar is easily found: get the sugar jar and add the sugar. The refinement of these abstract operators is sped up by providing macros for often recurring subsequences, e.g., fetching a jar or opening a jar. DoLittle's search depth is 2 operators only.

As can be seen in the previous example, no single problem solving strategy (macros, cases, abstractions) was able to solve the problem efficiently, but a combination had to be used.

2 PLANNING PARADIGM

This work is based on the plan-space search paradigm; planning is seen as search through possible plans. A planner P maintains a set of possible candidates C for successful plans. A possible candidate c is selected from this set. If the plan is a solution it is returned. Otherwise a plan transformation t is applied to the candidate to yield a new set of candidates.

In this framework, a planner is defined by the following three components:

1. The representation language for evolving plans. As a minimum, a plan language must be able to express a set of operators, an ordering on the operators, and a set of constraints on variable instantiations. Common plan languages are totally ordered, partially ordered, fully instantiated, or partially instantiated plans.

2. The set of plan transformations. A plan transformation t is a function that takes a plan expression and returns a new candidate plan.

3. The plan selection method. The selection method determines which candidate plan is expanded next from the candidate set. Popular plan selection methods are breadth-first, depth-first, and best-first.

The following paragraphs identify some popular planning strategies in this framework. An extensive

list can be found in [1].

Means-ends analysis planning maintains *two* sets of totally ordered, fully instantiated operators, a plan head and a plan tail. The plan head contains applied operators, whereas the plan tail contains operators that have not been applied yet. The set of plan transformations contains three plan transformations: (a) append a plan to the plan head, (b) prepend a plan to the plan tail, and (c) apply the first operator of the plan tail, which moves it at the end of the plan head.

Macro-based planning is a generalization of means-ends analysis planning; sequences of operators instead of single operators can be inserted into the plan head and tail. The plan language and the set of transformations from means-ends analysis planning are generalized to include sequences of operators.

Abstraction-based planning converts a problem domain into different levels of abstractions by assigning criticality levels to different literals. The plan language and the set of plan transformations are extended to support different criticality levels. Literals at a lower criticality level are ignored.

Case-based planning uses the same plan description language as means-ends analysis planning. However, it uses a larger set of plan transformations, such as insertion of an operator, removal of an operator, reordering of operators, replacement of an operator, and replacement of a variable binding.

This section showed that the plan space paradigm leads to a practical definition of planning strategies (a plan language and a set of plan transformations) and that very different planning strategies can be defined in this framework.

3 MULTI-STRATEGY PLANNER

This section discusses the design of DoLittle, a multi-strategy planner that can combine a wide variety of different planning strategies on a single problem. The set of planning strategies that DoLittle currently supports includes: forward chaining, means-ends analysis [2], case-based [3], automatic subgoaling, abstraction-based [4], and macro-based planning.

Extending the analysis of different planning strategies in the plan space paradigm described in section 2, the plan language and the set of plan transformations necessary for a multi-strategy planning system can be determined. In particular, the plan language must be able to represent: (a) totally ordered operator sequences, (b) instantiated variables, (c) a plan skeleton, and (d) trees of problem spaces. The set of plan transformations must include (a) operator transformations (application, insertion, removal, reordering, replacement of an operator sequence), (b) changing a variable binding, and (c) the creation of different subproblem spaces.

Design of a multi-strategy planner requires two key components: (a) a representation for different planning strategies and conditions under which they should be applied, and (b) a search strategy that emulates different planning strategies.

3.1 DOLITTLE'S REPRESENTATION

DoLittle uses *general operators*, a generalization of standard Strips like operators ([2]), to represent different planning strategies.

A general operator consists of sets of preconditions, open goals, and effects. Although general operators are syntactically similar to Strips operators, their semantics are very different. Pre-conditions and effects of a Strips operator define the applicability conditions of the operator, that is when *can* the operator be applied. DoLittle uses the preconditions and effects identify planner states (current state and unachieved goals) in which the operator *should* be applied.

Associated with a general operator is a set of refinements. A refinement is a sequence of general or primitive operators that guarantees that the effects of the parent operator are achieved, but it may have additional pre-conditions and effects.

The following general operator is an example from the kitchen domain and illustrates the key features:

```
GEN-PICK-UP-FROM-CUPBOARD
  Variables $OBJECT
  Preconds (ARM-EMPTY)
           (IS-AT ROBBY AT-TABLE)
           (IS-IN $OBJECT CUPBOARD)
  Open goals (HOLDING $OBJECT)
  Effects (HOLDING $OBJECT)
          (NOT (IS-IN $OBJECT CUPBOARD))
          (NOT (ARM-EMPTY))
  Refine. 1 PICK-UP-FROM-CUPBOARD($OBJECT)
  Refine. 2 OPEN-DOOR(CUPBOARD)
            PICK-UP-FROM-CUPBOARD($OBJECT)
```

The general operator GEN-PICK-UP-FROM-CUPBOARD can be used to pick up an object from the cupboard independent of whether the cupboard is open or not. The preconditions and open goals define the applicability conditions by specifying under which planner states, this general operator may be applied. The refinements are only applicable if the current world state matches them, i.e., the arm is empty, the robot is at the table, and $OBJECT is in the cupboard. Furthermore, the planner must be trying to achieve (HOLDING $OBJECT). The general operator has two refinements. Refinement 1 is used if the cupboard door is open, refinement 2 if the door is closed.

3.2 DOLITTLE'S SEARCH CONTROL

The representation of different planning strategies is alone not sufficient for a multi-strategy planning system. The representation of cases and macros are very similar, both are sequences of operators. However, their effect on the search space is very different. Macros are simply selected and concatenated, whereas cases are selected based on a similarity metric and are adapted. The situation is similar to that of asking for the inherent meaning of a bit-pattern (e.g., 11101010), which of course depends on whether it is interpreted as a binary number (signed or unsigned?), a machine code instruction (for which processor?), a character string, or a floating point number. Therefore, a multi-strategy planning system must also provide a search control method that emulates the effect of a given planning strategy on the search space.

DoLittle's search control method is based on a strongest-first heuristic: planning strategies that result in the smallest search space are tried first. Checking a small search space first has two benefits: (a) if a solution exists in this space, it can be found quickly, and (b) if no solution exists, the failure can quickly be recognized. Note that since planning strategies are not complete, they may remove the part of the search space with the solution.

DoLittle uses a domain independent similarity measure to retrieve the most similar general operator to the current problem. If the operator represents a macro that exactly matches a problem, a solution is found (macro-based planning). Otherwise, the operator sequence is adapted to the current situation (case-based). If there is no case or macro available, a subproblem search space is created. At present, there are three different types of search spaces in DoLittle: an abstract subgoal space, a serial subgoal space, and a general subgoal. The different search spaces represent different constraints on the search. For example, an abstract search space does not allow any plan that would change any higher level literals.

4 EVALUATION

This section discusses briefly the results of an evaluation of DoLittle's performance on a set of problem domains. For a detailed description of the methodology and the statistical analysis, please refer to [1]. The evaluation included three domains: (a) the blocks-world, (b) the towers of Hanoi, and (c) the kitchen domain.

The goal of this evaluation was to evaluate empirically the performance of multi-strategy planning (DoLittle) against that of four single strategy planners: a means-ends analysis planner, a case-based planner, a macro-based planner, and an abstraction-based planner. For the evaluation, a set of 250 test problems was generated and the performance on the test problems was measured.

An interesting aspect brought out by the empirical evaluation is that no single strategy planner performed consistently better than the other ones. For example, although case-based planning lead to the biggest improvement in the kitchen domain, it performed worth in the towers of Hanoi domain.

Multi-strategy planning performed slightly better than the single strategy planners in the towers of Hanoi domain (about 10% faster than abstraction-based). In the blocks-world and kitchen domain, multi-strategy planning performed much better than the single strategy planners (100-300% faster than case-based).

5 CONCLUSIONS

So far, the main focus of the research has been on methods for combining different planning strategies on a single problem. Current work investigates the interaction of different planning strategies.

The uniform representation in DoLittle also makes it an ideal test-bed for the comparison of different planning strategies. This work may result in a better understanding of the applicability conditions of different planning strategies. This will lead to better methods of determining when a given planning strategy is suitable for a domain.

Development on DoLittle is continuing at the University of Auckland. Currently, we are working on a project that uses DoLittle as the strategic planning component of an autonomous, mobile robot. The intended applications for the robot are mail delivery in an office environment, and other household and security tasks.

REFERENCES

[1] J. Baltes. *DoLittle: A learning multi-strategy planner.* PhD thesis, University of Calgary, Calgary, Canada, June 1996.

[2] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(2), 1972.

[3] K. J. Hammond. *Case Based Planning.* Academic Press Inc., 1989.

[4] C. A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[5] P. Stone, M. Veloso, and J. Blythe. The need for different domain-independent heuristics. In K. Hammond, editor, *Proceedings of the second international conference on artificial intelligence planning systems*, pages 164–169, Menlo Park, 1994. AAAI Press.