Why Plan Generation and Plan Execution Are Inseparable

Craig A. Knoblock
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
Email: knoblock@isi.edu

Introduction

People usually think of the planning problem as the problem of identifying a sequence of actions that when executed will achieve a goal. However, the purpose of planning is not actually the plan, but the achievement of the goals through the use of the plan. Perhaps a more appropriate formulation of the planning problem to think of it as the problem of achieving a goal through the execution of a sequence of actions. The issue is that due to the complexities and uncertainties in the real world it is rarely the case that one can simply formulate a plan and then execute it with guaranteed success. Instead, planning may require executing actions to gain additional information (i.e., sensing) or recovering from action failures due to uncertainty in the world or an incomplete world model.

In previous papers I described a planner called Sage that has been applied to the problem of information gathering (Knoblock 1994; 1995; 1996). The Sage planner supports simultaneous action execution, tightly integrates planing and execution, and provides replanning, planning for asynchronous goals, and sensing in a unified framework. The execution model is based on the execution framework that was developed for IPEM (Ambros-Ingerson 1987). Sage extended this framework to support simultaneous execution and planning in parallel with the action execution and is then applied to the information gathering task.

Information gathering task requires selecting, integrating, and retrieving data from distributed and heterogeneous information sources in order to satisfy a query. The relevant data must be selected from numerous, possibly overlapping or replicated sources. Integrating the information may be costly, especially when combining data from different sites. Retrieving the information may be time consuming due to the distribution of data and the contention for limited resources. Integration of plan generation and execution for this task results in a much more flexible and robust solution.

The remainder of this paper first reviews the integration of planning and execution in Sage and then examines some of the issues that arise in both generating executable plans and executing those plans for the information gathering task.

Integrating Planning and Execution

We have developed a planner called Sage that builds on the UCPOP partial-order planner (Barrett *et al.* 1993). UCPOP provides an expressive operator language that includes conjunction, negation, disjunction, existential and universal quantifiers, conditional effects, and a functional interface that allows preconditions to be implemented as Lisp functions. We extended this planner to support simultaneous action execution and to tightly integrate planning and execution.

Partial-order planners, such as UCPOP, produce plans with actions that are unordered. However, if two actions are left unordered they can be executed in either order, but not simultaneously. To execute actions in parallel in a partial-order planner requires that (1) actions can be executed simultaneously without changing the outcome of the individual actions, and (2) any potential resource conflicts must be captured in the representation of the operators in order to avoid conflicts during execution. We assume that the first condition holds (as it does in the information gathering domain) and we extended the planner to support the second condition. To support reasoning about resources, we added an explicit resource declaration to the action language, which describes the resources required when executing an action. We also augmented the planner to identify and remove potential resource conflicts. With these extensions, any actions left unordered in the final plan can be executed simultaneously.

Planning and execution are tightly integrated by considering execution as an integral part of the planning process. This is done by treating the execution of each individual action as a necessary step in completing a plan. The goal of the planner becomes producing a complete and executed plan rather than just producing a complete plan. Just as achieving all of the preconditions of a plan is required for a complete plan, executing each of the actions is also part of the final result.

Sage keeps track of the current status of every action in the plan by marking them as either *unexecuted*, *executing*, *completed*, or *failed*. This is similar to how execution was integrated into IPEM (Ambros-Ingerson 1987). The underlying planner, UCPOP, maintains a list of *flaws*, which is an agenda of things that need to be done to complete a particular plan. These flaws include *open conditions*, which are subgoals that have not yet been achieved, and *threats*, which are potential interactions between operators that must be resolved by adding ordering or binding constraints. We integrated execution in Sage by adding two new types of flaws: an *unexecuted* action flaw and an *executing* action flaw. Whenever a new operator is added to a plan, the corresponding flaw indicating that the action is unexecuted is also added to the agenda. The *executing* flaw is used to handle the fact that actions are not instantaneous and in some cases may take considerable time. A plan is not complete until all *unexecuted* and *executing* flaws have been removed.

The choice of when to execute an action in a plan is important, since undoing an executed action may be costly or impossible. An action cannot be executed until every precondition of the action has been both planned and achieved by executing the preceding actions. Even after an action is executable, Sage delays execution as long as possible to avoid committing to a partially constructed plan prematurely. Once an action has been executed, it is viewed as a commitment to the plan in which the action occurs – the planner cannot consider any plans that are not refinements of the plan being executed. The idea is that the planner should find the best complete plan before any action is executed. Then once execution is initiated, it resolves any failed subplans or new goals before executing the next action. This means that the planner will never execute an action until the corresponding plan is selected as the best available.

Since executing an action may take considerable time, the planner cannot simply execute an action and wait for the results. Instead, Sage creates a subprocess that executes the action and notifies the planner once it has completed. In order to keep track of the actions currently being executed, the corresponding *unexecuted* flaw is removed from the agenda and the *executing* flaw is added. At any one time there may be a number of actions that are all executing simultaneously. On each cycle of the planner, the system checks if any executing actions have completed. Once an action is completed, the *executing* flaw is removed from the agenda. If it completes successfully, the action is marked as completed. Other actions that depend on this action may now be executable if all of the other preceding actions have also been executed. If an action fails, the failed portion of the plan is removed and then replanned.

Sage's top-level algorithm for tightly integrating planning and execution is summarized in Table 1. The planner starts with an initial plan, where the goals are the open conditions. Initially, the set of *current plans* contains only this initial plan. It repeats the algorithm until it produces a plan in which every action has been executed. The planner considers only refinements of the *current plans*. Whenever an action is executed, an action terminates, or a new goal is added, the set of *current plans* is replaced by a new set containing only this new plan. The first two conditions in this algorithm ensure that the planner finds a plan with no open conditions or threats before it commits to a plan and initiates any actions.

Remove a plan from the set of *current plans* and apply the first applicable condition:

- If there are any threats, resolve them by adding additional constraints to the plan. Add the possible refinements to the *current plans*.
- If there are any open conditions, add additional actions or ordering links to achieve them. Add the possible refinements to the *current plans*. (Open conditions that contain run-time variables for sensing will be postponed.)
- If any executing actions have completed:
  - If the action completed successfully, record the results and update the plan. If the plan is complete, return the results. Otherwise, replace the *current plans* with this new plan.
  - If the action failed, remove the failed portion of the plan, update the model to avoid generating the same plan again, and replace the *current plans* with this new plan.
- If there are any new goals to solve, add them to the open conditions and replace the *current plans* with this new plan.
- If any unexecuted actions are now executable, create a process to execute them and replace the *current plans* with this new plan.

Table 1: Algorithm for planning and execution

This algorithm supports simultaneous planning and execution. Before the system initiates execution of any action, it constructs an initially complete plan. However, once execution starts, an action could fail, a new goal could arise, or the system may require additional information (sensing) to continue planning. In any of these cases, once the new open condition has been added to the list of flaws, the system can augment the executing plan to achieve these conditions while it continues executing any actions that have already been initiated.

The Nature of Executable Plans

An important aspect of an executable plan is the ability to exploit parallelism. A number of partial-order planners have exploited the partial-order nature of the plan to provide explicit parallelism in the execution.

However, the classes of parallel plans produced by these different planners differs due to the workings of the underlying planner (Knoblock 1994). It is interesting to note that many of these planners will both underconstrain the final plans in some ways and overconstrain them in others.

A partial-order planner may underconstrain a plan in at least two ways when unordered actions are executed in parallel. First, if resource constraints are not made explicit then actions could be executed in parallel that may conflict on their resource requirements. This can be addressed by making resource conflicts explicit as was done in SIPE (Wilkins 1984) or Sage (Knoblock 1995). Second, if the simultaneous execution of two actions changes the effects of either action, this may lead to a failure in later actions that expected certain conditions to be true. To address this problem requires either defining the actions such that this does not occur or using a richer representation language to capture this type of interaction.

Partial-order planners may also overconstrain an executable plan by imposing ordering constraints in the process of resolving threats or resource conflicts. The problem with these ordering constraints is that since actions are not instantaneous, it may be difficult to know at plan construction time which ordering to impose to avoid a conflict and still minimize the overall parallel execution time. As noted by Kambhampati et al. (1995) it is not strictly necessary to impose ordering constraints to resolve threats. Instead the planner can maintain the set of threats and verify that a plan consistent with the possible ordering constraints exists. This would allow the planner to order the actions at execution time based on the actual execution time of the preceding actions.

How Plans Should be Executed

A number of planners have treated execution as a separate problem. The system first formulates a plan and then sends it to a plan execution module to carry out the plan. In completely predictable and deterministic environments, this may work fine. However, both problems and opportunities may arise at execution time that can be resolved or exploited by tightly integrating the executor with a planner. In some cases a loose coupling of the planner and executor may be sufficient, but in the case where the planner is executing actions in parallel, it may be possible to refine plans in which other actions continue to be executed (Knoblock 1995). Sage does this in replanning failed actions and planning for asynchronous goals.

Another reason for tightly integrating the planning and execution is to exploit sensing as part of the planning process. Sensing can be used to gather additional information, optimize a plan, or in some cases provide missing information that is required to complete a plan. Sensing can also be used to help reduce the uncertainly in the planning process. Treating the execution of actions as part of the planning problem provides a much more natural mechanism for integrating sensing actions into a planner.

Discussion

Much of the work on planning make a number of very strong assumptions. Many planners assume that the world is deterministic, all of the preconditions and effects of an action are known, action execution is instantaneous, the planner has a complete model of the world, etc. In fact, when executing plans in real domains some or all of these assumptions will not hold. By considering the execution as an integral part of the planning problem, it will force researchers to find ways to address these assumptions, which will lead to the development of more practical planning and execution systems.

References

Ambros-Ingerson, J. 1987. *IPEM: Integrated Planning, Execution, and Monitoring.* Ph.D. Dissertation, Department of Computer Science, University of Essex.

Barrett, A.; Golden, K.; Penberthy, S.; and Weld, D. 1993. UCPOP user's manual (version 2.0). Technical Report 93-09-06, Department of Computer Science and Engineering, University of Washington.

Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: A unified framework for evaluating the design tradeoffs in partial order planning. *Artificial Intelligence* 76(1-2).

Knoblock, C. A. 1994. Generating parallel execution plans with a partial-order planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems.*

Knoblock, C. A. 1995. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence.*

Knoblock, C. A. 1996. Building a planner for information gathering: A report from the trenches. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems.*

Wilkins, D. E. 1984. Domain-independent planning: Representation and plan generation. *Artificial Intelligence* 22(3):269–301.