Planning

Michail G. Lagoudakis and Sven Koenig

Georgia Institute of Technology and University of Southern California

Planning is the process of generating a plan, that is, a course of action whose execution achieves a given objective. Planning is a key component of intelligent behavior. It often makes sense to support human planning or even automate it to find plans of high quality or find them fast because it often is difficult for humans to plan in complex situations. Examples include scheduling fleets of cargo planes or finding safe paths for Mars rovers, where the communication delay makes their teleoperation from Earth extremely tedious. Therefore, since the mid-1960s researchers, mostly from the artificial intelligence community, have studied how computers can plan, including how to classify planning problems, how to represent them formally, how to solve them, and how to integrate the resulting systems into human-machine environments. One deployed planning system is the Remote Agent, which autonomously controlled NASA's Deep Space 1 space probe during an experiement in 1999. (We mention the names of some other planning systems, mostly research prototypes, in brackets.)

Classification of Planning Problems

Suppose that we are interested in scheduling a fleet of planes to transport cargo between airports. We are given the current locations of all planes and cargo pieces and the destinations of all cargo pieces. We are asked to provide a plan for delivering all cargo pieces to their destinations. How can this air-cargo problem be formalized?

A *planning domain* is described by states and actions. *States* are snapshots of the world that include all of its aspects that are relevant for planning. In our case the states include the locations of all planes and cargo pieces. *Actions* are the means for changing the world. Their execution transforms the world from one state to another one. In our case actions include loading a cargo piece into a plane, flying a plane from one airport to another, and unloading a cargo piece from a plane. One needs to specify for each action and state

whether the action can be executed in the state and, if so, which state results from its execution. For example, a cargo piece can only be loaded into a plane if they are both at the same airport. In this case the cargo piece changes location from being on the ground to being in the plane.

A *planning problem* is described by the planning domain, the start state, and the planning objective. The planning objective is often given in the form of a *goal*, that is, a desired property of the world. The purpose of planning then is to determine a plan that transforms the start state into a goal state, that is, a state that satisfies the goal. These plans are called *valid plans*.

The characteristics of planning problems determine how one should represent them formally, how plans look like, how to find these plans, and how long it takes to find them. *Classical planning problems* assume that one can predict with certainty which state will result from an action execution (*deterministic actions*) and that states can always be completely and accurately observed (*totally observable states*). In general, plans can be arbitrary computer programs, but for classical planning problems, it is sufficient to consider only sequences of actions. We treat the air-cargo problem as a classical planning problem. Doing so is a simplification since, in the real world, airport workers sometimes make mistakes and load cargo pieces into the wrong planes, resulting in nondeterministic actions. In such cases one needs to search for the cargo pieces since their locations are not totally observable, resulting in only partially observable states.

Representations of Planning Problems

A *representation language* allows one to specify planning problems in a way that can be understood by a computer. More expressive representation languages allow one to model a greater variety of planning problems but are also more complicated, which makes it harder for humans to encode planning problems and understand planning problems that have been encoded by others, and also harder for computers to solve encoded planning problems. Researchers originally tried to use logic as representation language and theorem-proving techniques to find valid plans by proving that there exists an action sequence that transforms the start state to a goal state. However, logic is very expressive and theorem proving thus turned

out to be slow. Researchers therefore devised specialized representation languages for planning problems. One of the earliest such languages was used by a planning system called the Stanford Research Institute Problem Solver [STRIPS] in the early 1970s to control the Shakey robot. The STRIPS representation language is still widely used today.

STRIPS specifies states using a subset of logic, namely as conjunctions of propositions, that is, as sets of statements that are either true or false. For example, "At(C1,JFK)" denotes that cargo piece C1 is on the ground at airport JFK. Listed propositions are assumed to be true while unlisted propositions are assumed to be false (*closed-world assumption*). STRIPS specifies actions compactly using *action schemata*, that is, parameterized descriptions of actions that describe in which states the actions can be executed and which states result from their execution. An action is obtained from an action schema by supplying objects for its parameters. An action can be executed in all states that contain the propositions on its PRECONDITION list. Its execution then results in the state that is obtained from the state before its execution by deleting all propositions on its DELETE list and adding all propositions on its ADD list. The ADD and DELETE lists address the so-called *frame problem*, that is, how to specify the consequences of action executions efficiently, by not listing those propositions that remain unaffected by action executions. A simple version of the air-cargo problem can be described as follows in STRIPS:

Start State:

IsAirport(JFK), IsAirport(SFO), IsPlane(P1), IsPlane(P2), IsCargo(C1),

IsCargo(C2), At(C1,SFO), At(C2,JFK), At(P1,JFK), At(P2,JFK)

Goal:

At(C1,JFK), At(C2,SFO)

Action Schemata:

Load(cargo,plane,airport)

PRECONDITION LIST: IsAirport(airport), IsPlane(plane), IsCargo(cargo), At(plane,airport), At(cargo,airport)

DELETE LIST: At(cargo,airport)

ADD LIST: In(cargo,plane)

Unload(cargo,plane,airport)

PRECONDITION LIST: IsAirport(airport), IsPlane(plane), IsCargo(cargo), At(plane,airport), In(cargo,plane)

DELETE LIST: In(cargo,plane)

ADD LIST: At(cargo,airport)

Fly(plane,from_airport,to_airport)

PRECONDITION LIST: IsAirport(from_airport), IsAirport(to_airport), IsPlane(plane), At(plane,from_airport)

DELETE LIST: At(plane,from_airport)

ADD LIST: At(plane,to_airport)

The start state specifies that there are two airports (JFK and SFO), two planes (P1 and P2) and two cargo pieces (C1 and C2). Cargo piece C1 is on the ground at airport SFO, while cargo piece C2, plane P1, and plane P2 are on the ground at airport JFK. The goal specifies that C1 is to be transported to JFK and C2 to SFO; every state in which C1 is at JFK and C2 is at SFO is a goal state. The action schema "Load (cargo,plane,airport)" is for loading "cargo" into "plane" at "airport," where "cargo," "plane," and "airport" are parameters. The action schema "Unload(cargo,plane,airport)" is for unloading "cargo" from "plane" at "airport." Finally, the action schema "Fly(plane,from_airport,to_airport)" is for flying "plane" from "from_airport" to "to_airport." For example, to be able to load "cargo" into "plane" at "airport," it must be the case that "airport" is an airport, "plane" is a plane, "cargo" is a cargo piece, and both "cargo" and "plane" are at "airport." "Load(C2,P1,JFK)" is an action for loading cargo piece C2 into plane P1 at airport JFK. This action can be obtained from the action schema "Load(cargo,plane,airport)." It can be executed in the start state since the start state contains "IsAirport(JFK)," "IsPlane(P1)," "IsCargo(C2)," "At(P1,JFK)," and "At(C2,JFK)." Its execution in the start state results in the state

IsAirport(JFK), IsAirport(SFO), IsPlane(P1), IsPlane(P2), IsCargo(C1), IsCargo(C2),
At(C1,SFO), In(C2,P1), At(P1,JFK), At(P2,JFK)

Note that the STRIPS representation of the air-cargo problem correctly moves a cargo piece in a plane with the plane as it flies from one airport to another one. Thus, our STRIPS representation addresses the so-called *ramification problem*, that is, how to model the implicit consequences of action executions.

There are several valid action sequences for the air-cargo problem, including

Load(C2,P2,JFK), Fly(P2,JFK,SFO), Unload(C2,P2,SFO), Load(C1,P2,SFO),
Fly(P2,SFO,JFK), Unload(C1,P2,JFK)

The expressive power of STRIPS is limited. For example, it cannot express the consumption of continuous resources (such as fuel and time). Representation languages such as the Action Description Language (ADL) and the Problem Domain Description Language (PDDL) have therefore extended STRIPS to make it more expressive.

Solving Planning Problems

Planning techniques find valid plans for given planning problems. Planning can be understood as graph search, either in the state space or in the plan space. *State-space planning* searches the directed graph whose vertices correspond to states and whose edges correspond to state transformations, that is, actions. The objective of state-space planning is to find a path from the vertex that corresponds to the start state to a vertex that corresponds to a goal state. *Plan-space planning*, on the other hand, searches the directed graph whose vertices correspond to (possibly incomplete) plans and whose edges correspond to plan transformations. A plan is incomplete and thus invalid, for example, either if it is missing actions or if its actions are not completely ordered (*partial-order planning*) [NOAH, NONLIN, REPOP, SIPE, SNLP, TWEAK, UCPOP]. Partially ordered plans avoid unnecessary and potentially wrong ordering commitments between actions during planning (*least-commitment planning*). Plan transformations

therefore often add actions or ordering constraints between actions to plans. The objective is to find a path from the vertex that corresponds to the empty plan to a vertex that corresponds to a valid plan.

The search of both state-space and plan-space planning can proceed, for example, forward from the start vertex to the goal vertices (*progression planning*) or backward from the goal vertices to the start vertex (*regression planning*). The size of the search spaces often increases exponentially in the size of the planning problems. This implies that the graphs of typical STRIPS planning problems do not fit into the memories of computers and that finding shortest action sequences is computationally very hard (to be precise: PSPACE-complete). Fortunately, one can often find longer action sequences efficiently. One could, for example, use only one plane to transport all cargo pieces (one after the other) between the airports for the air-cargo problem, which results in unreasonably long action sequences for large planning problems. Planning techniques therefore exploit the structure of planning problems in an attempt to find reasonably short action sequences for realistically sized planning problems.

One way to exploit structure is to focus the search with heuristics, often in the form of distance estimates to the goal vertices [HSP, UNPOP]. If, for example, a certain number of airports still have cargo pieces on the ground that are not yet at their correct destinations, then one still has to execute at least twice that many actions, that is, a load action and an unload action for each such cargo piece. (We did not count fly actions since planes can carry several cargo pieces at a time.) Good distance estimates can be obtained automatically for state-space planning, for example, from a data structure called a planning graph [FF, GRAPHPLAN, IPP, LPG, SGP, STAN, TGP].

A second way to exploit structure is to use knowledge about the structure of valid plans [TLPLAN]. For example, one can ignore plans for the air-cargo problem that first load a cargo piece into a plane and then immediately unload it again.

A third way to exploit structure is to decompose planning problems into several subproblems that can be solved almost independently. Decomposing planning problems does not work well for puzzles like

the Rubik's Cube but seems to work well in domains in which humans plan well (*everyday planning*). For example, one can first identify the propositions that are part of the goal but not the start state, then find an action sequence for each of them that transforms the start state into a state that contains the proposition, and finally merge the actions sequences. This is the main idea behind *means-ends analysis*, which picks each of the propositions in turn and first finds an action whose add list contains the proposition and then recursively tries to achieve all of those preconditions of the action that do not already hold in the start state. One can also decompose planning problems hierarchically by refining high-level actions to make them more concrete (*hierarchical planning*) [ABSTRIPS, DEVISER, FORBIN, NONLIN, O-PLAN, SHOP, SIPE]. For example, once one has found a valid action sequence for the air-cargo problem, one can decompose the load actions into the various activities needed for loading a cargo piece into a plane, such as loading it into a van, driving the van to the plane, unloading it from the van, driving a ramp with a moving belt to the plane, activating the ramp, putting the cargo piece on the belt, and so on.

Finally, a fourth way to exploit structure is to speed up planning for the current planning problem by utilizing information about how one has solved similar planning problems in the past (*replanning* or *plan reuse*). Case-based planning, for example, adapts plans from similar planning problems in the past to fit the current planning problem [CAPER, CAPLAN, CHEF].

A planning technique is called *domain-dependent* if it needs to be provided with the structure of planning problems [PARCPLAN, TLPLAN]. It is called *domain-independent* if it discovers the structure of planning problems from their description [NONLIN, PRODIGY, SOAR, UCPOP]. A large number of novel domain-independent planning techniques were developed in the mid-1990s by using ideas from outside of artificial intelligence planning. For example, researchers found ways of mapping planning problems to heuristic search problems [FF, GRAPHPLAN, HSP, UNPOP], Boolean satisfiability problems [SATPLAN], integer-programming problems [SATPLAN-IP, STATE-CHANGE-IP], and model-checking

problems from hardware verification [HSCP, MIPS]. These problems can then be solved efficiently with known techniques, and a plan can easily be read of from those solutions.

Note that one can make mistakes or omit important facts when specifying planning problems. In such cases a plan that appears to be valid might not achieve the goal. It is thus important to monitor the execution of plans and either replan or repair them if they do not perform as expected [ASPEN, PLANEX]. One can also use techniques from machine learning to improve the specification of the planning problems automatically from observations made during plan execution [LPLAN].

More Complex Planning Problems

Which planning technique to use depends on the characteristics of the planning problem, the structure of the desired plans, and the planning objective. So far, we have described planning techniques for classical planning problems described in STRIPS. These planning techniques find valid plans in the form of action sequences, ideally sequences of small lengths. Researchers also study far more complex planning problems, for example, where conditions have to be maintained rather than achieved, where the world changes even if the planning system does not execute actions, and where other systems are present in either cooperative or competitive situations.

Uncertainty about the outcomes of action executions or the current state gives rise to decision-theoretic planning problems. In case of nondeterministic actions, a number of successor states can result from their execution (for example, due to actuator noise or not modeling all relevant aspects of the world) and one cannot predict in advance which one will actually result. In case of states that are not totally partially observable, some relevant aspect of the world cannot always be observed (for example, due to sensor limitations) or cannot always be observed correctly (for example, due to sensor noise). In the presence of nondeterministic actions or partially observable states, planning systems cannot always know their current states but can estimate them, for example, in the form of sets of possible states or probability distributions over them. Planning systems then often find plans that achieve the goal with high probability

or, if the goal can be achieved for sure, minimize the number of action executions either in the worst case or on average. For the objective of minimizing the number of action executions in the worst case, planning techniques can draw on ideas from artificial intelligence search. For the objective of minimizing the number of action executions on average, planning techniques can draw on dynamic programming ideas from operations research (such as value iteration and policy iteration) to solve totally and partially observable *Markov decision processes*, which generalize graphs. Totally observable Markov decision processes can model nondeterministic actions, while partially observable Markov decision processes can also model partially observable states. Decision-theoretic planning is computationally very hard, and decision-theoretic planning techniques thus exploit the structure of planning problems again, often by generalizing ideas from classical planning. For example, BURIDAN extends partial-order planning, SGP extends GRAPHPLAN, and MAXPLAN extends SATPLAN.

Some planning systems rely on coercion to achieve the goal without sensing and thus continue to find action sequences (*conformant or open-loop planning*) [CGP]. In general, however, it is not guaranteed that valid conformant plans exist for decision-theoretic planning problems or are of good quality because observations can now provide information about the current state. In general, one therefore often wants to find plans that contain sensing actions and select actions depending on the observations made during plan execution (*conditional, contingent, or closed-loop planning*) [WARPLAN-C]. For Markov decision processes it is sufficient to consider only conditional plans that map each state (for totally observable Markov decision processes) or each probability distribution over the states (for partially observable Markov decision processes) to the action that should be executed in it (*policies*), which reduces the search space and allows one to represent conditional plans compactly (*reactive planning*).

The large number of possible contingencies makes planning for decision-theoretic planning problems extremely time consuming. One way of speeding up planning is to interleave planning and action executions [CONT-PLAN, SEP-PLAN] since action executions can result in additional observations,

which can eliminate some contingencies and thus speed up planning. For example, instead of having to plan for all states that can result from the execution of an action, one can simply execute the action and then plan only for the successor state that actually resulted from its execution. There are different ways of interleaving planning and action executions. For example, *agent-centered planning techniques* find only the beginning of a valid plan by searching with a limited look-ahead around the current state, execute the plan, and repeat the process. *Assumption-based planning techniques*, on the other hand, find a plan that is valid provided their assumptions about the outcomes of action executions are correct. If these assumptions turn out not to hold during plan execution, they replan and repeat the process.

Planning and Human-Computer Interaction

Research on planning benefits from research on Human-Computer Interaction, and vice versa. A series of human computer interface issues need to get solved to deploy planning systems. For example, one needs to develop user interfaces for the acquisition of planning domains from experts and for the communication of the planning results to the users, especially for planning systems with human operators in the loop, such as decision-support systems and mixed initiative planning systems. Conversely, planning has been used to solve some Human-Computer Interaction tasks in software engineering and user-interface design. For example, NALIGE generates plans of low-level operating system commands to accomplish objectives specified by users via voice input. ASAP automates some aspects of software specification by proposing specifications, analyzing them to identify problems, and modifying them to resolve problems. CRITTER supports the interactive design of multiple interacting software systems. PALE extracts plan structure from program code, and ZIPPY then uses this knowledge to generate new code for an abstract specification. Finally, UCPOP and IPP have been used to automatically generate test cases for graphical user interfaces along with embedded verification steps.

Outlook

Planning is an exciting area of research that is important for manufacturing, space systems, disaster relief, robotics, defense, logistics, education, crisis response, entertainment, software engineering, and Human-Computer Interaction. The day when everyone has access to personalized planning systems is still to come, but planning technology is steadily improving. It is important to realize that aspects of planning and, more generally, decision making have been studied in various disciplines, including control theory, economics, operations research, decision theory, combinatorial optimization, and cognitive science<M>in addition to artificial intelligence. It is therefore interesting to combine ideas from different disciplines to build even more powerful planning systems. International planning competitions have been held since 1998 to compare the performance of planning systems on identical sets of planning problems and stimulate research on extending their capabilities in beneficial directions.

Michail G. Lagoudakis and Sven Koenig

Further Reading

Allen, J., Hendler, J., & Tate, A. (Eds.). (1990). *Readings in planning*. San Mateo, CA: Morgan Kaufmann.

Amant, R. (1999). User interface affordances in a planning representation. *Human-Computer Interaction, 14*(3), 317<N>354.

Anderson, J., & Fickas, S. (1989). A proposed perspective shift: Viewing specification design as a planning problem. *ACM SIGSOFT Software Engineering Notes, 14*(3), 177<N>184.

Blythe, J. (1999). Decision-theoretic planning. *AI Magazine*, *20*(2), 37<N>54.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research, 11*, 1<N>94.

Dean, T., & Wellman, M. (1991). *Planning and control.* San Mateo, CA: Morgan Kaufmann.

Fickas, S., & Helm, B. (1992). Knowledge representation and reasoning in the design of composite systems. *IEEE Transactions on Software Engineering, 18*(6), 470<N>482.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*(3,4), 189<N>208.

Fikes, R., & Nilsson, N. (1993). STRIPS: A retrospective. *Artificial Intelligence, 59*(1,2), 227<N>232.

Howe, A., von Mayrhauser, A., & Mraz, R. (1997). Test case generation as an AI planning problem. *Automated Software Engineering, 4*(1), 77<N>106.

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*, 99<N>134.

Koenig, S. (2001). Agent-centered search. *AI Magazine, 22*(4), 109<N>131.

Manaris, B., & Dominick, W. (1993). NALIGE: A user interface management system for the development of natural language interfaces. *International Journal of Man-Machine Studies, 38*(6), 891<N>921.

Memon, A., Pollack, M., & Soffa, M. (1999). Using a goal-driven approach to generate test cases for GUIs. *Proceedings of the International Conference on Software Engineering*, 257<N>266.

Rist, R. (1995). Program structure and design. *Cognitive Science, 19*, 507<N>562.

Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine, 15*(4), 27<N>61.

Weld, D. (1999). Recent advances in AI planning. *AI Magazine, 20*(2), 93<N>122.

Yang, Q. (1997). Intelligent planning: A decomposition and abstraction based approach. New York: Springer-Verlag.

Young, R., & Simon, T. (1988). Planning in the context of Human-Computer Interaction. *Proceedings of the Conference of the British Computer Society Human-Computer Interaction Specialist Group*, 363<N>370.