Assimilating Planning Domain Knowledge from Other Agents

Tim Grant

Netherlands Defence Academy
P.O. Box 90.002
4800 PA Breda, Netherlands
tj.grant@nlda.nl / tgrant@cs.up.ac.za

Abstract

Mainstream research in planning assumes that input information is complete and correct. There are branches of research into plan generation with incomplete planning problems and with incomplete domain models. Approaches include gaining knowledge aimed at making the input information complete or building robust planners that can generate plans despite the incompleteness of the input. This paper addresses planning with complete and correct input information, but where the domain models are distributed over multiple agents. The emphasis is on domain model acquisition, i.e. the first approach. The research reported here adopts the view that the agents must share knowledge if planning is to succeed. This implies that a recipient must be able to assimilate the shared knowledge with its own. An algorithm for inducing domain models from example domain states is presented. The paper shows how the algorithm can be applied to knowledge assimilation and discusses the choice of representation for knowledge sharing. The algorithm has been implemented and applied successfully to eight domains. For knowledge assimilation it has been applied to date just to the blocks world.

Introduction

The plan generation process takes as its input a planning problem consisting of initial and goal states and a domain model typically consisting of planning operators. Its output is a sequence of actions – a plan - that will, on execution, transform the initial state to the goal state.

To locate the research reported here, we place the planning process into its wider context. In Figure 1 the Planning process is central. Its output – a plan – is ingested by the Controlling process. In executing the plan, the Controlling process issues commands to the Process Under Control (PUC), and receives sensory information back.

The Planning process itself has inputs: the domain model and the initial and goal states. The usual assumption is that these inputs come directly from the Controlling process. However, we take the view that each input is developed by an intervening process: initial states result from State Estimation[1], goal states from Goal Setting, and domain models from Modelling. It is these three processes that receive feedback from the Controlling process in the form of the observed sensory information. State Estimation uses the feedback to identify the PUC's current state. Goal Setting determines whether the current goal state has been achieved, can be maintained, or must be replaced by another goal state. Modelling assesses whether the domain model remains a complete and correct description. If not, it uses the feedback to modify or extend the domain model. This paper centres on the Modelling process.

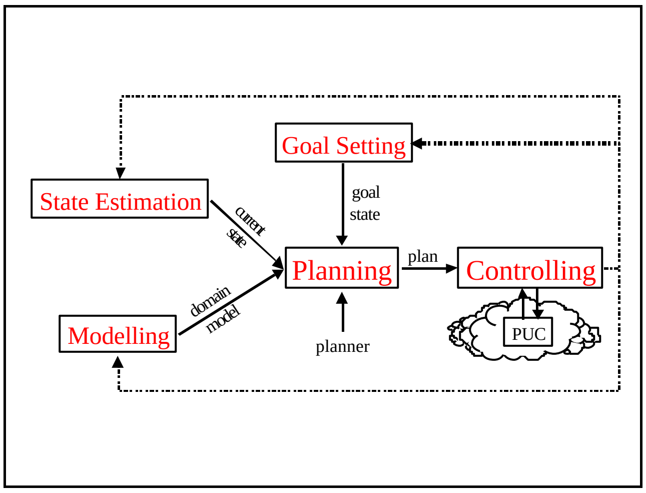

Figure 1. Planning in context.

Mainstream research in planning assumes that the input information is complete and correct[2]. In practical applications, however, information about the domain model, the planning problem, or both may be incomplete and/or incorrect. In the literature there are two approaches to planning with incomplete and/or incorrect input information (Garland & Lesh, 2002):

- Gain better information, either during plan generation or during plan execution. This may be done by using sensors embedded in the PUC to acquire information, by consulting an oracle (e.g. an expert), or by trial-and-error learning from performing experiments in the domain. The acquired information may be used in state estimation, in goal setting, and/or in modelling.
- Build robust planners that can generate plans that succeed regardless of the incompleteness and/or

[1] The term is borrowed from the process control literature.

[2] By convention, the goal state is usually a formula describing a set of (goal) states.

incorrectness of the input information. *Conformant planning* (Goldman & Boddy, 1996) is planning with incomplete knowledge about the initial state and/or the effects of actions. *Model-lite planning* (Kambhampati, 2007) is planning with an incomplete or evolving domain model. *Erroneous planning* (Grant, 2001) has the more limited aim of characterizing the types of erroneous plans generated if the planner is not robust (the error *phenotypes*), based on concepts drawn from the literature on human error, and trying to understand the causes for the observed errors (the error *genotypes*). Knowledge of the error phenotypes and genotypes could then be used for *plan repair* (Krogt, 2005).

By contrast, this paper is concerned about planning with complete and correct input information, but where that information is distributed across multiple agents. In particular, it is concerned with distributed domain models. While the domain model is complete for some set of agents, each individual agent's domain model is (initially) incomplete.

This paper adopts the view that, where knowledge about the planning domain is distributed over multiple agents, the agents must share that knowledge if planning is to succeed. To do so, they must be interoperable. The source of the knowledge and its recipient must adopt a common knowledge representation, as well as coordinating their knowledge-sharing actions. Moreover, the recipient must be capable of assimilating the knowledge gained (Lefkowitz & Lesser, 1988) into other knowledge it may already have. Assimilation of another agent's domain model is an extension of the Modelling process. This paper focuses on the knowledge assimilation capability and choosing a suitable representation for knowledge sharing.

The subject matter in this paper touches on several theoretical areas. Firstly, it is based on the application of machine learning to planning, because knowledge assimilation is a learning process. More specifically, it is concerned with applying machine learning techniques to the acquisition of planning operators. Secondly, because the recipient's domain model is evolving, it touches on model-lite and erroneous planning. Thirdly, it is based on communication theory and, in particular, on information or knowledge sharing concepts drawn from management and organization theory.

The paper is divided into seven chapters. Chapter 2 describes the author's algorithm for modelling planning domains by acquiring planning operators from example domain states. Chapter 3 introduces knowledge sharing based on the Shannon (1948) model of communication. Chapter 4 describes the assimilation of planning domain knowledge, and chooses a suitable representation for sharing that knowledge between agents. Chapter 5 describes two simple worked examples. Chapter 6 surveys related research. Finally, Chapter 7 draws conclusions, identifying the key contributions of this paper, its limitations, and where further research is needed.

Modelling Planning Domains

The author's algorithm for modelling planning domains by acquiring planning operators from example domain states is known as Planning Operator Induction (POI) (Grant, 1996). As the name indicates, POI employs inductive learning from examples. More specifically, it embeds Mitchell's (1982) *version space and candidate elimination* algorithm, taking selected domain states as input examples and inducing STRIPS-style planning operators.

The POI algorithm has been implemented and applied successfully to eight domains (Grant, 1996), including the blocks world, the dining philosophers problem, and a model of a real-world spacecraft payload based on a chemical laboratory instrument. For knowledge assimilation it has been applied to date just to the blocks world.

The ontology employed in POI separates the domain representation into static and dynamic parts. The static part of the POI ontology represents invariant domain entities in terms of object-classes and -instances, of inter-object relationships, and of inter-relationship constraints. By convention, relationships and constraints are binary[3]. For example, the blocks world consists of Hand, Block, and Table object-classes. The `holding` relationship links an instance of the Hand (object-) class to an instance of the Block class, and the `onTable` relationship links a Block instance to a Table instance. The `holding` and `onTable` relationships are constrained in that no Block instance may be both held and on the table simultaneously. Such constraints are known in the planning literature as *domain axioms* or *invariants* and in the database literature as *cardinality* and *exclusion constraints* (Nijssen & Halpin, 1989). The static part of the ontology (less the exclusion constraints) may be depicted using Chen's (1976) Entity-Relationship Diagramming (ERD) notation[4].

The dynamic part of the POI ontology represents domain entity behaviour in terms of states, transitions, and planning operators. Planning operators are reformulations of classes of domain transitions. Instantiated relationships synchronise the states of objects. For example, the `holding hand1 block2` relationship synchronises the states of the objects `hand1` and `block2`: `hand1` must be holding `block2` and `block2` must be held by `hand1` simultaneously. If `hand1` ceases to be holding `block2`, then `block2` must simultaneously cease being held by `hand1`. Transitions combine synchronised changes in relationships. For example, the cessation of the

[3] Higher arity relationships and constraints can be reduced to binary relationships and constraints by changing how the object - and relationship -classes, respectively, are modelled. Details are given in Grant (1996).

[4] The ERD notation is limited to depicting constraints between two instances of the same relationship, i.e. cardinality constraints. It cannot depict constraints between instances of two different relationships, i.e. exclusion constraints. The POI ontology and algorithm is not so limited.

`holding hand1 block2` relationship may be combined with the advent of the `onTable block2 table1` relationship. In terms of Allen's (1983) temporal logic, we would say that the relationships *meet*. Note that they meet because the domain constraint from the previous paragraph forbids them from overlapping. Grant (1995) says that the transition *pivots* around the (instantiated) binary domain constraint.

The POI ontology is similar to McCluskey and Porteus' (1997) object-centred representation for the specification of planning domains. The key differences are that, in POI, the relationships and constraints are strictly binary. Moreover, the constraints hold only between relationships. In addition, objects, relationships, constraints, states, and transitions all have classes, i.e. sorts in McCluskey and Porteus' terminology.

The POI algorithm has two parts:

- *Part 1: Acquisition*. The purpose of the first part of POI is to acquire a static, object-oriented model of the domain from example domain states. POI does not require that the example domain states form a valid sequence, plan-segment, or plan, unlike other algorithms for acquiring planning operators. However, the examples may have to be carefully chosen. Part 1 subdivides into three steps:
  - *Step 1.1*: Acquire domain state description(s).
  - *Step 1.2*: Recognise the objects and relationships in the state description(s).
  - *Step 1.3*: Compile cardinality and exclusion constraints from the objects and relationships. The constraints can be generated exhaustively by constructing all possible pairs between relationships that share an object. For example, pairing the relationship `holding ?Hand1 ?Block1` with `holding ?Hand1 ?Block2` expresses the domain constraint that a hand cannot hold two (or more) blocks simultaneously[5]. By default, a constraint is assumed to hold if no counterexample can be found among the acquired domain state descriptions. Thus, if an agent observed a domain state in which two hands were indeed holding the same block then this constraint would no longer hold.
- *Part 2: Induction*. The purpose of the second part of POI is to induce a dynamic model of domain behaviour from the static, object-oriented domain model. Domain behaviour is modelled using a state-transition network from which planning operators can be extracted. Part 2 sub-divides into six steps:
  - *Step 2.1*: Generate the description language for the domain. The description language is the set of all relationships between object-instances that satisfy the cardinality and exclusion constraints.
  - *Step 2.2*: Construct the version space for the description language using the cardinality and exclusion constraints to eliminate invalid candidate nodes. The version space is a partial lattice of valid nodes, with each node being described in terms of relationships between the domain object-instances.
  - *Step 2.3*: Extract the domain states from the version space. The domain states are the lattice nodes in the maximally specific boundary of the version space.
  - *Step 2.4*: Using the Single Actor / Single State-Change (SA/SSC) meta-heuristic, determine the domain transitions between the domain states. The SA/SSC heuristic is that a single object (the actor) initiates the transition, undergoing a change in just one of its relationships. The actor is at the root of a causal hierarchy of state-changes in the other participating objects. For example, in the blocks world when a robot hand picks up a block from the table, the hand is the actor, making true its `holding` relationship with the block being picked up. The hand's action causes the block both to act on itself so that it is no longer `clear` and to act on the table, breaking the `onTable` relationship.
  - *Step 2.5*: Generalise the domain transitions as transition-classes.
  - *Step 2.6*: Reformat the transition-classes as planning operators.

Depending on how POI is to be used, Part 1 may be optional. If an agent observes an existing domain and uses POI to gain knowledge about how to plan actions in that domain, then Part 1 is essential. By contrast, if an ERD or equivalent static model of a domain (which may not yet exist) is available, then modelling can proceed directly to Part 2. In knowledge assimilation, one agent (the *source*) performs Part 1 and another (the *recipient*) performs Part 2.

Knowledge Sharing

Information and knowledge sharing has been extensively studied in management and organization theory. For simplicity, we will take the terms "information" and "knowledge" as being interchangeable, *pace* Ackoff (1989). Information sharing is a dyadic exchange of information between a source and a recipient (adapted from Szulanski (1996), p.28). Sharing involves the dual problem of "searching for (looking for and identifying) and transferring (moving and incorporating) knowledge across organizational subunits" (Hansen, 1999, p.83). For the purposes of this paper, we will take knowledge sharing as meaning knowledge transfer. Searching for or discovery of other agents that have suitable complementary knowledge about a domain is an area for future research.

Shannon's (1948) model of communication is useful for thinking about knowledge sharing. In the Shannon model, the source and recipient each operate within their own organizational contexts. Information transfer begins when the source generates a message. The message is encoded into a form (a signal) in which it is transmitted by means of a communications medium, such as electromagnetic waves, telephone cables, optical fibres, or a transportable electronic storage medium. Random noise and systematic distortion may be added during

[5] By convention, the same instance of the Block object-class cannot be matched to the two different variables `?Block1` and `?Block2`.

transmission. The recipient decodes the signal and assimilates the decoded message into its own store of knowledge.

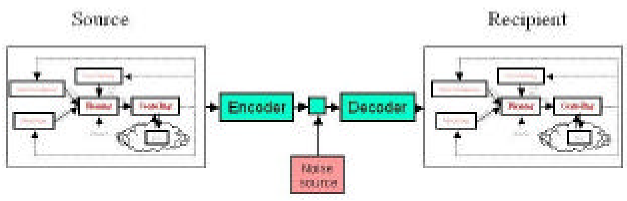

Figure 2. Linking source and recipient agents using Shannon (1948) model.

For the purposes of this paper, we assume that the source and recipient are agents with an internal structure as shown in Figure 1. In general the agents should be able to exchange the outputs of their respective Planning, Controlling, State Estimation, Goal Setting, and Modelling processes, given suitable encoders and decoders (Figure 2). We concentrate here on the Modelling process, how the source's knowledge should be encoded, and what decoder the recipient needs to assimilate that knowledge. We neglect the issue of noise and distortion in this paper.

Assimilating Planning Domain Knowledge

Lefkowitz and Lesser (1988) discuss knowledge assimilation in the context of acquiring domain knowledge from human experts. Their implemented system, $K^{n}_{A}c$, was developed to assist experts in the construction of knowledge bases using a frame-like representation. Assimilated knowledge represented domain objects, relationships, and events. The main contribution of their research was in developing several generic techniques for matching sets of entities and collections of constraints. Research questions included:

- How does the expert's domain description correlate with the description contained in the knowledge base?
- How should the knowledge base be modified based on the expert's new information?
- What should be done when the expert's description differs from the existing one?

Despite the contextual differences, there are strong parallels between Lefkowitz and Lesser's (1988) work and assimilating planning domain knowledge. Assimilation of domain knowledge should be integrated with plan generation and execution. It should permit a variety of ways of learning, including *learning-by-seeing* (i.e. by observing the domain and inferring what actions are possible), *learning-by-being-told* (e.g. by domain experts or other agents), and *learning-by-doing* (i.e. by generating and executing plans). When knowledge is distributed over multiple agents, then individual agents may need to combine different ways of learning. In particular, an agent may well need to combine knowledge it gained from its own observations of a domain with information it has gained by being told by another agent. Like Lefkowitz and Lesser, learning concerns domain objects, relationships, constraints, and events. Analogues of Lefkowitz and Lesser's research questions apply; here we are concerned with the planning analogue of their second question.

Considering the POI algorithm from the viewpoint of encoding and decoding, we see that there are three forms in which knowledge relating to the domain model could be exchanged:

- *As cases*. The source agent could transmit the domain states it has observed, i.e. the input information to Part 1 of the POI algorithm. The source agent would not have to process its observations before transferring them to the recipient. The recipient agent would then have to add the source's domain states to its own database of domain states, and perform Parts 1 and 2 of the POI algorithm to obtain a set of planning operators. Exchanging knowledge in this form is likely to be verbose for real-world domains, possibly with duplicated observations. More importantly, it would limit knowledge assimilation to learning-by-seeing. The only thing that knowledge sharing achieves is that the recipient can "see" both what it can itself observe and what the source has observed.
- *As static domain models*. The source agent could transmit its static domain model, i.e. the information as output by Part 1 and as input to Part 2. The source agent would have had to perform Part 1 before transmitting its static domain model to the recipient. The recipient agent would then have to add the source's objects, relationships, and constraints to its own database of objects, relationships, and constraints. Where source and recipient agents disagree on whether a constraint holds, then the constraint is assumed not to hold (because one of the agents will have seen a counterexample). The recipient retains its own list of object-instances and does not assimilate the source's object-instances list, because the recipient may not be able to execute plans on objects that it cannot see. Then the recipient would perform Part 2 of the POI algorithm to obtain a set of planning operators. Exchanging knowledge in this form is likely to be concise. Moreover, it would allow learning-by-seeing, learning-by-being-told, and their combination.
- *As planning operators*. The source agent could transmit its dynamic domain model, i.e. the information as output by Part 2. The recipient agent would then simply have to add the planning operators obtained from the source to its own planning operators. Exchanging knowledge in this form is still more concise, but assumes that (1) the source and the recipient agents' observations are sufficiently rich for both of them to be able to induce a set of planning operators, and that (2) their sets of planning operators are complementary. There is no way for additional planning operators to be induced by synergy.

In this research, the encoding-decoding schema has been determined by the researcher. Ideally, the source and recipient agents should themselves be able to negotiate a suitable encoding-decoding schema, depending on considerations such as privacy, security, and communications bandwidth. Further research is needed to provide agents with such a capability.

Worked Examples

Two worked examples should make the key issues clear. The first example is the one-block world and the second is taken from the three-blocks world. Because the one-block world is simple, the first example is described in more detail. The second example illustrates the need to select example states carefully if the agents are to induce a full set of planning operators.

Suppose two agents each observe a different state of a one-block world (Slaney & Thiebaux, 2001), as represented by Nilsson (1980)[6]. There are two possible states[7]: [`[holding hand1 block1]` `[onTable block1 nil]` `[onTable nil table1]`] and [`[holding hand1 nil]` `[holding nil block1]` `[onTable block1 table1]`]. Let us suppose that Agent1 is given the first state description and Agent2 the second.

The following table depicts the static domain model that would result from their performing Part 1 separately, i.e. without knowledge sharing and assimilation:

	Agent1's model	**Agent2's model**
Object-classes	Hand, Block, Table	Hand, Block, Table
Object-instances	hand1, block1, table1	hand1, block1, table1
Relations	holding ?Hand ?Block onTable ?Block nil onTable nil ?Table	holding ?Hand nil holding nil ?Block onTable ?Block ?Table
Constraints	IF holding ?Hand1 ?Block1 AND holding ?Hand1 ?Block2 THEN INVALID -- hand cannot hold multiple blocks IF holding ?Hand1 ?Block1 AND holding ?Hand2 ?Block1 THEN INVALID -- block cannot be held by multiple hands IF holding ?Hand1 ?Block1 AND onTable ?Block1 nil THEN INVALID -- block cannot be held and not on a table NOTE: This constraint does not hold because the observed state is a counterexample. IF onTable ?Block1 nil AND onTable ?Block2 nil THEN INVALID -- multiple blocks cannot be off the table IF onTable nil ?Table1 AND onTable nil ?Table2 THEN INVALID -- multiple tables cannot be clear at same time	IF holding ?Hand1 nil AND holding ?Hand2 nil THEN INVALID -- multiple hands cannot be empty at same time IF holding nil ?Block1 AND holding nil ?Block2 THEN INVALID -- multiple blocks cannot be not held IF holding nil ?Block1 AND onTable ?Block1 ?Table1 THEN INVALID -- block cannot be not held and on a table NOTE: This constraint does not hold because the observed state is a counterexample. IF onTable ?Block1 ?Table1 AND onTable ?Block1 ?Table2 THEN INVALID -- block cannot be on multiple tables IF onTable ?Block1 ?Table1 AND onTable ?Block2 ?Table1 THEN INVALID -- table cannot hold multiple blocks

Neither of the agents would be able to induce any planning operators, because POI Part 2 would simply result in the induction of a single state, namely the state each agent had observed originally. There needs to be a minimum of two states for the SA/SSC heuristic to find any transitions.

Now suppose that the agents share their domain knowledge. Since neither of them can induce planning operators separately, exchanging data in the form of planning operators is not feasible. However, they can exchange knowledge in the form either of cases or of their static domain models. For the one-block world it is simpler for the agents to exchange cases, but this does not apply to complex, real-world examples.

Sharing their domain models enables the agents to create synergistic knowledge. Firstly, Agent1 learns from Agent2 that blocks can be on tables, and Agent2 learns from Agent1 that hands can hold blocks. Secondly, additional constraints can be identified, as shown in the following table:

	Synergistic knowledge
Relations	holding ?Hand ?Block holding ?Hand nil holding nil ?Block onTable ?Block ?Table onTable ?Block nil onTable nil ?Table

Constraints	IF holding ?Hand1 nil
	AND holding ?Hand1 ?Block1
	THEN INVALID
	-- hand cannot be both empty and holding a block
	
	IF holding nil ?Block1
	AND holding ?Hand1 ?Block1
	THEN INVALID
	-- hand cannot be both held by a hand and not held
	
	IF holding ?Hand1 ?Block1
	AND onTable ?Block1 ?Table1
	THEN INVALID
	-- block cannot be both held and on a table
	
	IF onTable nil ?Table1

[6] Distinguishing three object-classes (`Hand`, `Block`, `Table`) and yielding four operators (`pickup`, `putdown`, `stack`, `unstack`). See Grant et al, 1994.

[7] A third state would be observed in an orbiting spacecraft: [`[holding hand1 nil]` `[holding nil block1]` `[onTable block1 nil]` `[onTable nil table1]`]. During development of the POI algorithm the three states were indeed induced, resulting in the induction of a set of six operators (`pickup, putdown, floatoff, floaton, letgo, capture`). The author observed that he had failed to represent the action of gravity. To do so while retaining the Nilsson (1980) domain representation requires a triple constraint, stating in effect that a block must be either held by a hand or supported by a table or by another block. This can be solved by extending the POI ontology, either by allowing constraints of arity higher than two or by introducing an inheritance hierarchy of object-classes. The author adopted the latter solution, because this has the synergistic consequence of reducing the complexity of the version space, leading to savings in induction time and memory requirements (Grant, 1996).

	AND `onTable` ?Block1 ?Table1 THEN INVALID -- table cannot be supporting both a block and nothing IF `onTable` ?Block1 nil AND `onTable` ?Block1 ?Table1 THEN INVALID -- block cannot be both off and supported by a table

The synergistic knowledge, together with the additional constraints, enables the agents to induce the `pickup` and `putdown` planning operators. They do not have enough knowledge to induce the `stack` and `unstack` operators because stacks of blocks and the `on` relationship between blocks does not exist in the one-block world.

The three-blocks world has 22 states, falling into five state-classes (Grant et al, 1994). Experiments with the implemented POI algorithm, adapted for knowledge assimilation, showed that it is not necessary for the agents to observe all 22 states (Grant, 1996). Just two, judiciously-chosen, example states sufficed[8]. In one state the hand must be empty, and in the other it must be holding a block. One state must show a stack of at least two blocks, and one stack must show two or more blocks on the table. Inspection shows that there are four pairings of the five state-classes that can meet these requirements. Two can be rejected on the grounds that they are *adjacent*, i.e. that they are separated by the application of just one operator. Successful knowledge assimilation has been demonstrated for the remaining two state-pairs: for all three blocks on the table paired with the state in which one block is held and the other two are stacked, and for a stack of three blocks paired with the state in which one block is held and the other two are on the table. Moreover, the induced set of planning operators can be used to generate and successfully execute a plan that passes through at least one novel state, i.e. a state that the agents had not previously observed.

It is not known whether two (judiciously-chosen) example states suffice in all domains for the induction of a full set of planning operators. Hand simulations and experiments have only been done for the (one-hand, one-table, and) one- and three-blocks worlds. More research is needed, e.g. by applying knowledge assimilation using POI to the International Planning Competition benchmark domains and to real-world domains where planning knowledge is distributed geographically or organizationally.

Related Work

In 2003, Zimmerman and Kambhampati surveyed the research on applying machine learning to planning. They identified three opportunities for learning: before planning, during planning, and during execution. Learning techniques applied fell into two groups: inductive versus deductive (or analytical) learning. Inductive techniques used included decision tree learning, neural network, inductive logic programming, and reinforcement learning. They observed that early research emphasised learning search control heuristics to speed up planning. This has fallen out of favour as faster planners have become available. There is now a trend towards learning or refining sets of planning operators to enable a planner to become effective with an incomplete domain model or in the presence of uncertainty. "Programming by demonstration" can be applied so that the user of an interactive planner could create plans for example problems that the learning system would then parse to learn aspects peculiar to the user.

In terms of Zimmerman and Kambhampati's (2003) survey, this paper applies Mitchell's (1982) inductive version space and candidate elimination algorithm to planning. The POI algorithm could be used before planning, during planning, or during execution. It centres on the learning of domain models in the form of planning operators. It exhibits an element of "programming by demonstration" in that the user shows POI example domain states, rather than example plans or execution traces.

In his 2006 lectures on learning and planning at the Machine Learning Summer School, Kambhampati distinguished three applications of learning to planning: learning search control rules and heuristics, learning domain models, and learning strategies. Research in learning domain models could be classified along three dimensions: the availability of information about intermediate states, the availability of partial action models, and interactive learning in the presence of humans. POI does not need information about intermediate states nor partial action models, and it does not require the presence of humans. By comparison, other operator learning algorithms require as input:.

- Background domain knowledge: Porter & Kibler (1986), Shen (1994), Levine & DeJong (2006).
- Partial domain model (i.e. operator refinement, rather than *ab initio* operator learning): Gil (1992), DesJardins (1994), McCluskey et al (2002).
- Example plans or traces: Oates and Cohen (1996), Wang (1996), Yong et al (2005).
- Input from human experts: McCluskey et al (2002). POI can accept a static domain model from a human expert (e.g. for a domain that does not yet exist) instead of observing domain states, but this is not applicable to assimilating domain knowledge distributed over multiple agents.

POI is closest to Mukherji and Schubert (2005) in that it takes state descriptions as input and discovers planning invariants. The differences are that POI also discovers objects and relationships and uses the information it has discovered to induce planning operators. Like McCluskey and his collaborators (McCluskey & Porteus, 1997; McCluskey et al, 2002), POI models domains in terms of object-classes (sorts, in McCluskey's terminology), relationships, and constraints.

[8] Introspection suggests that there may be a single state in the two-hands, four-block world that could provide all the information needed to induce all four operators, but then the knowledge could not be distributed over multiple agents.

Conclusions

This paper has addressed the topic of planning with a domain model that is complete and correct but distributed across multiple agents. The paper takes the view that the agents must share their knowledge if planning is to succeed. The Planning Operator Induction (POI) algorithm (Grant, 1996) has been introduced as a means of acquiring planning operators from carefully-chosen examples of domain states. Unlike other algorithms for acquiring planning operators (Porter & Kibler, 1986) (Gil, 1992) (Shen, 1994) (DesJardins, 1994) (Wang, 1996) (Oates & Cohen, 1996) (McCluskey et al, 2002) (Yang et al, 2005) (Mukherji & Schubert, 2005) (Levine & DeJong, 2006), the example domain states do not need to form a valid sequence, plan-segment, or plan, nor do preceding or succeeding transitions have to be given. When agents share their partial knowledge of the domain model, the two parts of the POI algorithm can be divided between the source and recipient in the knowledge-sharing process. The agents exchange the static, object-oriented domain model resulting from Part 1 of the POI algorithm. This enables the recipient to identify synergies between the shared knowledge and knowledge it already has and to perform the induction, i.e. Part 2 of the algorithm.

This paper makes several contributions. Its primary contribution is in showing how planning domain knowledge that is distributed across multiple agents may be assimilated by sharing partial domain models. Secondary contributions include:

- The POI domain-modelling algorithm is presented that acquires planning operators from example domain states. The example domain states do not need to form a valid sequence, plan-segment, or plan, nor do preceding or succeeding transitions have to be given.
- The ontology used in the POI algorithm extends McCluskey and Porteus' (1997) object-centred representation. Relationships and constraints are strictly binary. Constraints are between pairs of relationships, rather than domain-level axioms. Hence, both relationships and constraints are associated with (classes of) domain objects.

A key limitation of the research reported here is that, while knowledge assimilation using the POI algorithm has been implemented, it has only been tested for the (one-hand, one-table, and) one- and three-blocks worlds. Future research should include:

- Applying POI-based knowledge assimilation to a wider variety of planning domains, e.g. International Planning Competition benchmark domains. One research question to be addressed is whether two (judiciously-chosen) example states suffice in all domains for the induction of a full set of planning operators.
- Elucidating the conceptual links between the POI algorithm and plan generation using planning graphs.
- Applying the POI algorithm to sense-making, i.e. the modelling of novel situations (Weick, 1995). An approach has been outlined in Grant (2005).
- Extending the POI ontology to model inheritance and aggregation relationships, with the eventual aim of using the Unified Modeling Language (UML) as a representation for the static, object-oriented and dynamic, behavioural domain models in the POI algorithm.
- Developing an integrated planning environment that incorporates domain modelling, plan generation, plan execution, state estimation, and goal setting to act on real and simulated domains.
- Extending agent capability to (1) negotiating mutually-acceptable encoding-decoding schemes, and (2) discover agents that have complementary knowledge.
- Investigating the application of knowledge assimilation using POI to real-world domains where planning knowledge is distributed geographically or organizationally. Example domains include air traffic control and military Command & Control (Grant, 2006).

References

Ackoff, R. 1989. *From Data to Wisdom*. Journal of Applied Systems Analysis, 16, 3-9.

Allen, J.F. 1983. *Maintaining Knowledge about Temporal Intervals*. Communications of the ACM, 26, 11, 832-843.

Chen, P.P-S. 1976. *The Entity-Relationship Model: Towards a unified view of data*. ACM Transactions on Database Systems, 1, 9-36.

DesJardins, M. 1994. *Knowledge Development Methods for Planning Systems*. Proceedings, AAAI-94 Fall Symposium series, Planning and Learning: On to real applications. New Orleans, LA, USA.

Fikes, R., & Nilsson, N.J. 1971. *STRIPS: A new approach to the application of theorem proving to problem solving*. Artificial Intelligence Journal, 2, 189-208.

Garland, A., & Lesh, N. 2002. *Plan Evaluation with Incomplete Action Descriptions*. TR2002-05, Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts, USA.

Gil, Y. 1992. *Acquiring Domain Knowledge for Planning by Experimentation*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Goldman, R., & Boddy, M. 1996. *Expressive Planning and Explicit Knowledge*. Proceedings, AIPS-96, 110-117, AAAI Press.

Grant, T.J. 1995. *Generating Plans from a Domain Model*. Proceedings, 14th workshop of the UK Planning and Scheduling Special Interest Group, 22-23 November 1995, University of Essex, Colchester, UK.

Grant, T.J. 1996. *Inductive Learning of Knowledge-Based Planning Operators*. PhD thesis, University of Maastricht, The Netherlands.

Grant, T.J. 2001. *Towards a Taxonomy of Erroneous Planning*. Proceedings, 20th workshop of the UK Planning and Scheduling Special Interest Group, 13-14 December 2001, University of Edinburgh, Scotland.

Grant, T.J. 2005. *Integrating Sensemaking and Response using Planning Operator Induction*. In Van de Walle, B.

& Carlé, B. (eds.), Proceedings, 2nd International Conference on Information Systems for Crisis Response and Management (ISCRAM), Royal Flemish Academy of Science and the Arts, Brussels, Belgium, 18-20 April 2005. SCK.CEN and University of Tilburg, 89-96.

Grant, T.J. 2006. *Measuring the Potential Benefits of NCW: 9/11 as case study*. In Proceedings, 11th International Command & Control Research & Technology Symposium (ICCRTS06), Cambridge, UK, paper I-103.

Grant, T.J., Herik, H.J. van den, & Hudson, P.T.W. 1994. *Which Blocks World is* the *Blocks World?* Proceedings, 13th workshop of the UK Planning and Scheduling Special Interest Group, University of Strathclyde, Glasgow, Scotland.

Hansen, M. T. 1999. *The Search-Transfer Problem: The role of weak ties in sharing knowledge across organization subunits*. Administrative Science Quarterly, 44 (1), 82-111.

Kambhampati, S. 2006. *Lectures on Learning and Planning*. 2006 Machine Learning Summer School (MLSS'06), Canberra, Australia.

Kambhampati, S. 2007. *Model-lite Planning for the Web Age Masses: The challenges of planning with incomplete and evolving domain models*. Proceedings, American Association for Artificial Intelligence.

Krogt, R. van der. 2005. *Plan Repair in Single-Agent and Multi-Agent Systems*. PhD thesis, TRAIL Thesis-series T2005/18, TRAIL Research School, Netherlands.

Lefkowitz, L. S., and Lesser, V. R. 1988. *Knowledge Acquisition as Knowledge Assimilation*. International Journal of Man-Machine Studies, 29, 215-226.

Levine, G., & DeJong, G. 2006. *Explanation-Based Acquisition of Planning Operators*. Proceedings, ICAPS 2006.

McCluskey, T.L., & Porteus, J.M. 1997. *Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency*. Artificial Intelligence Journal, 95, 1-65.

McCluskey, T.L., Richardson, N.E., & Simpson, R.M. 2002. *An Interactive Method for Inducing Operator Descriptions*. Proceedings, ICAPS 2002.

Mitchell, T.M. 1982. *Generalization as Search*. Artificial Intelligence Journal, 18, 203-226.

Mukherji, P., & Schubert, L.K. 2005. *Discovering Planning Invariants as Anomalies in State Descriptions*. Proceedings, ICAPS 2005.

Nijssen, G.M., & Halpin, T.A. 1989. *Conceptual Schema and Relational Database Design: A fact-oriented approach*. Prentice-Hall Pty Ltd, Sydney, Australia.

Nilsson, N.J. 1980. *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, California, USA.

Oates, T., & Cohen, P.R. 1996. *Searching for Planning Operators with Context-Dependent and Probabilistic Effects*. Proceedings, AAAI, 865-868.

Porter, B., & Kibler, D. 1986. *Experimental Goal Regression: A method for learning problem-solving heuristics*. Machine Learning, 1, 249-284.

Shannon, C.E. 1948. *A Mathematical Theory of Communication*. Bell System Technical Journal, 27, 379-423 (July) & 623-646 (October).

Shen, W.-M. 1994. *Discovery as Autonomous Learning from the Environment*. Machine Learning, 12, 143-156.

Slaney, J., & Thiébaux, S. 2001. *Blocks World Revisited*. Artificial Intelligence Journal, 125, 119-153.

Szulanski, G. 1996. *Exploring Internal Stickiness: Impediments to the transfer of best practice within the firm*. Strategic Management Journal, 17, 27-43.

Wang, X. 1994. *Learning Planning Operators by Observation and Practice*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA.

Weick, K. 1995. *Sensemaking in Organizations*. Sage, Thousand Oaks, CA, USA. ISBN 0-8039-7178-1.

Yang, Q., Wu, K., & Jiang, Y. 2005. *Learning Action Models from Plan Examples with Incomplete Knowledge*. Proceedings, ICAPS 2005, 241-250.

Zimmerman, T., & Kambhampati, S. 2003. *Learning-Assisted Automated Planning: Looking back, taking stock, going forward*. AI magazine, 73-96 (Summer 2003).