Domain Analysis and Domain Knowledge: Generation, Representation, and Implementation

Ulrich Scholz
Intellectics Group
Darmstadt University of Technology
scholz@thispla.net

Abstract

Domain analysis is the inference of knowledge, termed domain knowledge, which is implicit in the description of planning problems. For many planning techniques, the use of domain knowledge yields substantial improvements of performance. This paper is concerned with domain analysis in three different ways. Firstly, we propose the domain knowledge exchange language DKEL as a mean to represent domain knowledge and to connect different planning techniques. We then present path reduction, a domain analysis technique, and lastly, we describe a novel approach to implement domain analysis techniques and demonstrate it by its use for path reduction.

Planning and Domain Analysis

One characteristic of planning is that planning problems can encode instances of several problem domains at the same time, *e.g.* a logistics problem and a TSP. Hereby, the difficulty of such problems lies in their combination rather than in the hardness of the respective partial problems. As a consequence, the design of general planning systems is not aimed at a specialization on a specific problem type.

Nevertheless, adaption to a considered planning domain and problem yields significant improvements of performance. The base for such an adaption is an examination of the problem prior to the actual search for a plan, a process we term domain analysis. The knowledge inferred by domain analysis is then used to adopt the planner, *i.e.*, to choose an appropriate planning technique and to transform the initial problem in a suitable representation.

Domain analysis techniques, and therefore the adoption of the planner, show a high variation. They can vary between a low level problem reduction, where the adaption needs or reveals almost no structure of the problem, and a high level decomposition, where subproblems are identified and solved by special purpose techniques. All these techniques have in common that they produce (and use) knowledge about the considered problem, hence it is of interest to study domain analysis independent of planning.

The work outlined in this paper is concerned with domain analysis in three different ways. Firstly, we propose the domain knowledge exchange language DKEL as mean to represent domain knowledge and to connect different planning techniques. We then present path reduction, a domain analysis technique which demonstrates the use of DKEL as input and output language. Lastly, we describe our implementation of path reduction. The chosen approach is interesting in its own right and can be applied to other preprocessing techniques as well.

DKEL: A Language to Represent and Exchange Domain Knowledge[1]

The knowledge input to a planning system may be divided in two distinct classes: problem specification and advice. The problem specification in turn typically consists of two parts: (1) a description of the means at the planners disposal, such as the possible actions that may be taken and resources that may be consumed, and (2) the goals to be achieved, including possibly a measure that should be optimized, constraints that should never be violated, and so on. Advice we term knowledge, of all kinds, intended to help the planner find a better solution, find it more quickly or even to find a solution at all.

In between specification and advice, a third class of knowledge, commonly called *domain knowledge,* may be distinguished. Briefly, it consists of statements about a planning problem that are logically implied by the problem specification, but that is not part of the specification. We would like to amend this definition with the requirement that domain knowledge is "planner independent", *i.e.* not closely tied to the internal workings of any particular planning system, but such a requirement is difficult to formulate precisely.

There are several good reasons for making this distinction. First of all, domain knowledge is implied by the problem specification, so it can be derived from same, and in many cases derived automatically. In this way it is different from advice, which must be provided by a domain expert, or learned from experience over many similar problems. There is a large, and growing, body of work on automatic domain analysis of this kind.

Furthermore, good planner advice tends to depend on knowledge both about properties of the problem and the planner used to solve it. For a given planner, there is often a fairly direct mapping from certain classes of domain

[1] DKEL is joint work with Patrik Halsum.

knowledge to useful advice for that planner. To take a simple example, in a regression planner an obvious use of state invariants is to cut branches of the search tree that violate an invariant. This is a sound principle, since a state that violates an invariant can never be reached and thus a goal set that violates the invariant is unreachable. The principle is founded on knowledge of how the planner works, but depends also on the existence of a certain kind of domain knowledge, namely state invariants.

On the other hand, domain knowledge in itself does not determine its use for advice. To continue the example, state invariants have many more uses: the MIPS planner uses them to find efficient state encodings (Edelkamp & Helmert 1999) and to find abstractions for generating heuristics (Edelkamp 2001), while in GRT (Refanidis & Vlahavas 2001) they are used to split the problem into parts and to improve the heuristic. In principle, the same planner can use the same domain knowledge in different, mutually exclusive ways.

In order to separate the generation and use of domain knowledge, we need some means of exchanging this knowledge between producer and consumer. What we propose is to "standardize" the expression of domain knowledge, using a language that builds on PDDL (McDermott *et al.* 1998), to make this exchange as natural and easy as the passing of a problem specification to a planner. In short, what PDDL has done for planning problem specification, we wish to do for domain knowledge. To this end, we have created the Domain Knowledge Exchange Language. The main goal of DKEL is to enable the kind of quick and easy prototyping of integrated planning systems outlined above. At the same time, it provides a limited taxonomy of different kinds of domain knowledge, with an attempt at a rigorous definition of the semantics for each kind.

DKEL augments the original domain or problem description with domain knowledge rather than altering or reducing it right away. Preserving the original structure of the domain and problem specification has several advantages: First of all, it is a prerequisite for the "chaining" of several analysis techniques described above. In addition, it leaves the choice of what knowledge to apply, and how to apply it, up to the planner. As mentioned earlier, turning domain knowledge into effective advice for a specific planner depends on knowledge of the workings of that planner, and including this in the exchange language would blur the separation between the generation and use of knowledge that it is meant to help achieve. Likewise, the decision what knowledge generating components to couple to a specific planner belongs with the user and not in the language that provides the coupling.

The explicit representation of domain knowledge also has other uses than simply connecting different parts of a planner. For example, it opens up the possibility of reasoning about the planning process. This use is demonstrated by the planner HAP (Vrakas, Tsoumakas, & Vlahavas 2002), whose planning strategy is adjusted according to the existence and characteristic of domain properties. Statements of domain knowledge, *e.g.* a state invariant, are regarded as property of the corresponding domain, similar to details like the number of goal facts.

There are also problems related to the use of domain knowledge in general and to the use of an explicit representation of it in particular: Not all knowledge is useful to all planners, and may even be detrimental if incorrectly used. Even if a particular item of knowledge is useful, the computational cost of inferring it may be higher than the benefit incurred by its use. But these problems are not intrinsic to explicit representations of domain knowledge, but only more prominent for them.

Problems related to the explicit representation of domain knowledge are the separation of domain analysis from its use in planning, which means that the planner component loses control over how knowledge is generated. Knowledge expressed in an interchange format may be incorrect due to a flawed analysis, differences in the interpretation of knowledge statements, or even sheer malice. However, adopting an explicit representation for domain knowledge, such as DKEL, may also help in avoiding such problems by offering a human-readable intermediate format with a well-defined semantics and by encouraging empirical evaluation of planning tools.

By now, DKEL has been subjected to relatively little in the way of evaluation. How does one evaluate a language, especially a language targeted at the role we have in mind for DKEL? While expressivity can be formally analyzed and compared, the most important metric of the value of DKEL is acceptance. Currently, there are three domain analysis tools that use DKEL as output language: Discoplan, TIM_dkel, a reimplementation of TIM, and RedOp. Furthermore, Varrentrapp *et al.* (2002) demonstrate the usefulness of DKEL with an on-line testbed for planning systems. DKEL is joint work with Patrik Haslum. A throughout discussion of the current development of DKEL has been presented at the ICAPS'03 workshop on PDDL (Haslum & Scholz 2003).

Path Reduction

Path reduction is a preprocessing technique aimed at reducing planning problems via the analysis of specific state invariants. In short, these state invariants can be seen as automata, and if a planning domain exhibits such invariants, a solution to a planning problem in that domain corresponds to a path for every automaton of its domain. In other words, every automaton accepts if it receives a solution as input. This allows to consider a solution of a planning problem as composition of separate solutions to the automata of the corresponding domain.

The view of planning domains as collection of automata suggests a planning strategy via divide and conquer: For each automata find a path that reaches its accepting state. Then combine these separate paths into one solution plan. Unfortunately, automata are not amenable to such a simple idea: The execution of a path in one automaton has side-effects on the other automata. Such side-effects complicate the composition or arbitrary paths to a conflict-free sequence. In fact, finding a path through an automaton that is part of a solution is at least as hard as planning itself. Instead of using automata for planning directly, we propose their use in a reduction technique.

The core idea of path reduction is to relate paths according to their replaceability in plans. If we consider the set of solutions to a given planning problem in respect to a single automation, the paths in that automaton are related across all these solutions. One similarity is, for example, that they all begin and end with the same facts, which are given by the initial state and the goal of the problem, respectively. We want to use this similarity to find and exclude paths that are unnecessary for solving the problem. A path P is obviously unnecessary if for every plan that uses P there is another plan that differs from the first solely by its use of another path instead of P. We call such paths P replaceable. We now examine paths regarding their replaceability and accept paths as relevant that cannot be replaced. We then exclude paths from solutions that are not identified as relevant.

What do we expect from path replaceability? In short, we want (1) that many paths are replaceable, (2) that replaceable paths can be exchanged in many plans, and (3) that we can easily test whether the exchange of a replaceable path is possible. These design goals are contradictory to some extend and they cannot be meet simultaneously in full. The first point states that the number of replaceable paths should be large, thus we have a large potential of reduction. The more similar we require replaceable paths to be, the less paths can be replaced. Therefore, we want to pose as few conditions on similarity as possible, *e.g.* we want to abstract from the middle parts of a path and only consider its beginning and end. The ideal of applicability, the second point, means that if we call two paths replaceable, we want the replacement to be valid in any plan, *i.e.* regardless of the context. Clearly, the less restrictions we pose on the definition of replaceability, the more likely it is that we need to test whether a replacement is valid or not. Point three states that this test should be easy.

Some properties of planning problems complicate the definition of path replaceability, especially properties that do not match the view of planning domains as collection of automata. Such properties include the deletion of all facts of a state invariant, which corresponds to an automaton without active state, and the existence of facts that are not element of any state invariant.

With a function to decide path replacement, we can build a constructive technique, whose application results in a set of paths that is guaranteed to suffice for a solution: path reduction. Commencing with an attempt to find a path from the initial state to the goal and by using the knowledge of the corresponding facts, we accept any path as relevant that connects these two facts and that is not replaceable by another path. We then identify fact pairs that have to be connected in order to execute these paths and identify relevant paths that connect these facts. We repeat this step until we find a fixed point. This algorithm, with some enhancements, guarantees that the fixed point is found quickly and that the paths in a fixed point suffice to find an optimal solution to the problem under examination.

Path reduction is similar to the reduction techniques like ROS (Scholz 1999) and RedOp (Haslum & Jonsson 2000), which compare action sequences and exclude some of them, too. Both ROS and RedOp on the one side and path reduction on the other side compare subsequences of plans. They differ in the kind of sequences under investigation, the class of candidate sequences, and how the results are used to reduce planning. In particular, ROS and RedOp operate on subsequences of plans, *i.e.*, conflict-free action sequences. If such a sequence is part of a plan, it is not interleaved by any other action of the plan. Their application results in a set of sequences that can be excluded from the search. Only few planning systems can use domain knowledge of this kind.

In contrast, path reduction operates on paths, which are not necessarily conflict-free and which can be interleaved by other actions in a plan. Thus, path reduction allows to compare sequences that cannot be compared by the others. The application of path reduction onto a planning problem yields another planning problem, which then can be solved by any planning system that could solve the initial one. After the fixed point of relevant paths is found, all actions that are not used by these paths are irrelevant. If a planning system uses this reduced action set, it does never consider paths that use these irrelevant actions. All actions of the reduced problem are actions of the original one, so a solution to the reduced problem is a solution to the original problem, too. One drawback of path reduction is its reliance on state invariants because it is not amenable to domains without. Fortunately, among the planning domains currently under investigation such domains are rare.

By now, we have examined several possible designs of path replacement. The formalism of path reduction has been worked out and a prototype of path reduction, based on a ground representation, shows reasonable performance. The final implementation is finished in parts and its completion is underway.

Implementing Domain Analysis Techniques: The General Case

PDDL allows to state planning problems with features of first order logic, *e.g.* it allows parameterized operators whose preconditions and effects are formulas. Even if these formulas are restricted to conjunctions of literals, they can still be rather complex to process. For example, the unification of two parameters with the same object can result in a formula that has several occurrences of the same ground predicate. Statements about combinations of operators, *e.g.* to infer statements of domain knowledge, further complicate the situation.

There are two common strategies to evade this problem: grounding and syntactic restrictions. After transforming a planning problem in a grounded form, operators and statements of domain knowledge can be represented by sets of ground literals. Consequently, operations on such a representation are basically set operations, which do not exhibit the aforementioned problems. The drawback of this approach is a substantial increase in problem size and a loss of explicit structure.

The second strategy is to restrict oneself to a syntactic subclass of all planning problems. For example, the planner IPP (Koehler *et al.* 1997) does not consider actions that result from unifying two parameters of an operator with the same constant. Hereby, it looses completeness but it never has to consider, *e.g.*, the unification of two literals in a precondition. Another example for such a syntactic restriction is to solely consider literals that co-occur in the precondition and effect of operators, *i.e.* the occurrence of a predicate in the precondition and in the effect of an operator such that both have the same parameter vector. This way, a planning technique would not consider literals that unify otherwise, even though these literal are part of a domain structure similar to those that the technique is aimed at. An example for a system that poses this kind of restriction is TIM (Fox & Long 1998).

There is no doubt that current planners which are based on grounding, like FF (Hoffmann & Nebel 2001), are capable of solving huge planning problems. Nevertheless, if we want to increase the size of problems further, we will eventually have to evade grounding. Likewise, tools that use syntactic restrictions yield excellent results, but these restrictions are somehow unsatisfactory and are not suitable for all techniques of interest. Consequently, we aim at a design of domain analysis tools that is as general as possible.

A likely consequence of implementing a domain analysis technique for the most general case is that we have to use theorem proving. Consider the (easy) question whether or not a given operator of a planning domain can always replace another in solution plans. This is true if for every ground instance of the second operator there is a corresponding ground instance of the first. Answering this question for the general case basically corresponds to proving a theorem.

Of course, reducing planning to theorem proving is not a practical approach. Instead, we use other computationally advantageous techniques, like constraint programming, to find candidate statements of domain knowledge. We then employ a theorem prover as subsystem to accept or reject these candidates. Hence, the considered theorems remain rather simple compared to the ones that would be necessary to find such candidates.

Currently, we use this approach for an implementation of the domain analysis technique path reduction, which has been described in the previous section. The implementation is partially finished and shows good performance.

Conclusions

In our view, domain analysis plays an increasing role in planning. The work described in this paper contributes to this area in three ways: We proposed the domain knowledge exchange language DKEL as a mean to represent domain knowledge and to connect planning tools. Then we presented path reduction, a domain analysis technique. Finally, we described a novel way to implement planning techniques in a general way, *i.e.* not based on grounding and independent of syntactic restrictions.

References

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In Fox, M., and Biundo, S., eds., *Proc. 5th European Conference on Planning*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 135–147. New York: Springer.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proc. 6th European Conference on Planning*, Lecture Notes in Artificial Intelligence. New York: Springer.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research* 9:367–421.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator sets. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proc. 5th Conference on Artificial Intelligence Planning & Scheduling*, 150–158. Breckenridge, USA: AAAI Press.

Haslum, P., and Scholz, U. 2003. Domain knowledge in planning: Representation and use. *ICAPS'03 Workshop on PDDL*. Trento, Italy.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In Steel, S., and Alami, R., eds., *Proc. 4th European Conference on Planning*, volume 1348 of *Lecture Notes in Computer Science*. Toulouse, France: Springer.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C. A.; Ram, A.; Veloso, M.; Weld, D.; and Wikins, D. 1998. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Refanidis, I., and Vlahavas, I. 2001. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research* 15:115–161.

Scholz, U. 1999. Action constraints for planning. In Biundo, S., and Fox, M., eds., *Proc. 5th European Conference on Planning*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 148–160. New York: Springer.

Varrentrapp, K.; Scholz, U.; and Duchstein, P. 2002. Design of a testbed for planning systems. In McCluskey, L., ed., *AIPS'02 Workshop on Knowledge Engeneering Tools and Techniques for AI Planning*, 51–58.

Vrakas, D.; Tsoumakas, G.; and Vlahavas, I. 2002. Towards adaptive heuristic planning through machine learning. In Grant, T., and Witteveen, C., eds., *UK Planning and Scheduling SIG Workshop*, 12–21.