Reducing Planning Complexity with Topological Abstraction

Adi Botea
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2E8
adib@cs.ualberta.ca

Abstract

Despite major progress in AI planning over the last few years, many interesting domains remain challenging for current planners. This document presents a model for reducing planning complexity by using topological abstraction. The model performs a clustering of the problem representation space, aiming to catch local relationships inside clusters and keep interactions between clusters as limited as possible. In effect, the initial problem is decomposed into a two level hierarchy of subproblems, each much simpler than the initial one. We present some application domains, pointing out the generality of the model. We also show the current status of our work and what we plan to achieve in the future.

Introduction

AI planning has recently achieved significant progress in both theoretical and practical aspects. The last few years have seen major advances in the performance of planning systems, in part stimulated by the planning competitions held as part of the AIPS series of conferences (McDermott 2000; Bacchus 2001; Fox & Long 2002). However, many hard domains still remain a great challenge for the current capabilities of planning systems.

Abstraction is a natural approach to simplify planning in complex problems. For instance, humans often create abstract plans that they try to follow during their search. In this paper we present *LAP* (Local Abstraction in Planning), a technique for reducing complexity of planning domains. Our approach abstracts the problem state representation, grouping related low-level features in local clusters. This type of abstraction is particularly useful for domains where the world is represented as a spatial structure such as a map or a maze. In such a domain, the clustering applies to atomic predicates describing the problem topology. This is the reason why we call this topological abstraction. To point out the generality of our approach, we describe a few application domains. We also show the current status of our work and further challenges that we plan to address in the future.

Motivation

Many interesting domains are hard to deal with when no abstraction is present. Examples of such domains are Sokoban and path-finding. In these domains, a hierarchical problem decomposition based on topological clustering can lead to significantly better performance. Our preliminary work using these domains as a testbed has already shown an impressive potential of the topological abstraction.

In domains such as Logistics, topological abstraction of the real world is part of the domain definition. In Logistics, several packages have to be transported from their initial location to various destinations. A Logistics problem has a map of cities connected by airline routes. Transportation inside cities can be done by truck (there is one truck in each city). Cities are abstracted, being treated as black boxes. Inside a city, a truck can go from any point to any destination at no cost. However, in the real world, a city is a graph of locations connected by streets and traveling inside a city involves considerable costs. In this context, removing human expertize and automatically obtaining abstracted models of the real world is an important research problem.

In Sokoban, which is a robotics application, a man in a maze has to push stones from their initial positions to designated destinations called *goal squares*. The maze contains inaccessible *wall* squares and accessible *interior* squares. There are two types of actions allowed. A *man movement* action changes the man's position to a free adjacent square. A *stone push* action changes the position of both the man and the considered stone. The preconditions request that the man is located next to the stone and the opposite position adjacent to the stone is a free interior square. The effects are that the stone has moved to the position that was initially free and the man has moved to the initial stone position. Both the AI planning and the single-agent search communities agree that this is a hard domain. The game is difficult for a computer for several reasons including deadlocks (positions from which no goal state can be reached), the large branching factor (can be over 100 – if we consider as moves all the stone pushes in the man reachable area), and long optimal solutions (can be over 600 moves). Another problem is that all known lower-bound heuristic estimators for the solution length are either of low quality, or expensive to compute.

Humans, who solve Sokoban puzzles much easier than state-of-the-art AI applications, abstract the maze into rooms and tunnels and use this high-level representation to create abstract plans. Following the humans' example, an AI application can cluster atomic squares into more abstract features such as rooms connected by tunnels, reducing the complexity of the hard initial problem. In effect, a large number of atomic squares is replaced by a few abstract, more meaningful features such as rooms and tunnels.

In the domain of path-finding, an agent on a map has to find a (shortest) path from its current position to a destination position. The map topology can have many forms, such as a battlefield, the interior of a building, etc. The problem is important in commercial computer games, robot planning, military applications, etc. The efficiency of the path-finding algorithms is often crucial, as they have to produce solutions in real-time and use limited resources (Yap 2002). The classical solving strategy represents the maze as a grid of atomic cells and uses a search algorithm such as A* on that graph. An action is to move to an adjacent cell that is not part of an obstacle. The representation of states in the search space greatly influences the efficiency of the search. A fine granularity of the map leads to a large search space, requiring serious time (and possibly space) resources. A much more efficient problem representation is to abstract the map into connected clusters such as rooms, large obstacle-free areas, bridges, etc. As in Sokoban, the abstract map representation is a small graph of connected clusters, with a much reduced search space.

The Model Overview

Abstraction is a powerful general technique for reducing problem complexity, and many different problem-solving strategies based on abstraction have been developed. The novelty of our LAP model is that it abstracts the state representation by exploiting topological relationships and forming boundaries between problem components. Atomic features that describe a state are grouped together into clusters that become state features in the new problem formulation. Since local properties can be handled more efficiently if separated from the global planning problem, the clustering aims to catch the local relationships and keep interactions between clusters as limited as possible.

Our approach abstracts the initial problem into a two level hierarchy of sub-problems, each being much simpler than the initial problem. We obtain a *global planning component* and a collection of local problems, called *local components*, aiming to exploit local relationships in the problem space and reuse local repetitions in the problem solution. In the global problem, the initial space is replaced by a much smaller abstract representation, which uses the states of local clusters to characterize the global abstract states. At this level, clusters are treated as black boxes, keeping their internal properties hidden. Each cluster has a local problem associated with it. Local problems do not interact directly. They only interact through the global level, thereby keeping the complexity of the model as low as possible.

For the global problem, we define abstract actions based on the local clusters that we use for abstract state description. We impose that an abstract action refers to either changing the internal state of one cluster or changing the internal state of two neighbor clusters, with no effects on the rest of the space. This is a powerful yet natural constraint as, in many

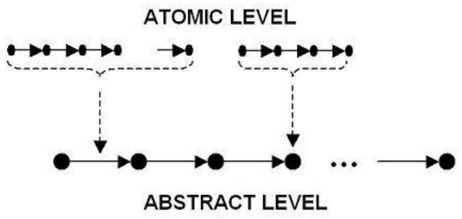

Figure 1: Abstraction reduces the size of a plan: (a) one abstract action contains several atomic actions; (b) when equivalent states are merged into one abstract state, actions that make transitions between the equivalent states are implicitly included in the abstract state.

real-life domains, the planning agent either does local processing or makes a transition from one local cluster to another. Examples of clusters can be rooms and corridors in a robot planning domain, cities on a map, large obstacles or obstacle-free areas in path-finding problems, etc.

The abstract planning problem constructs an abstract plan (i.e., a sequence of abstract actions) that can always be translated to a low-level solution. When an abstract plan is mapped to the initial problem, one abstract planning action is mapped to a macro operator, which is a sequence of atomic actions. The existence of such a mapping is ensured by our precondition checking mechanism. Checking the preconditions of an abstract action is equivalent to computing a corresponding sequence of atomic moves that respects all the low-level constraints in the current world. This computation constitutes the local component of our problem decomposition model.

Using abstract actions instead of atomic actions reduces both the branching factor and the depth of the global search space by collapsing a sub-tree into a single move. In addition to using abstract actions, the initial space is further reduced by identifying equivalent cluster states that can be merged into one abstract cluster state. Two cluster states s_1 and s_2 are equivalent if there exist two sequences of *local* atomic actions a_{12} and a_{21} so that the sequence a_{ij} $(i, j \in \{1, 2\})$ changes the state of the cluster from s_i to s_j. An action is local to a cluster if we can apply it without interacting with the exterior of the cluster. Sequences of local atomic moves which make transitions between equivalent states are "compiled away" completely and are not present at the global planning level at all. In effect, the planning problem becomes much easier, as the global search space is much smaller. Figure 1 illustrates how both macro moves and state equivalence reduce the complexity of an abstract plan as compared to the corresponding low-level solution.

In our approach, abstract solutions are correct, which means that they can always be mapped to a low-level solution. In contrast, the solution optimality is not preserved at the atomic level. If for each action of an optimal abstract solution we compute an optimal sequence of atomic moves, the resulting low-level solution is not guaranteed to be optimal. However, sub-optimal solutions are acceptable for many applications. For instance, Sokoban is so hard that any solution will do. If desired, non-optimal solutions can be improved in a post-processing phase. In path-finding, sub-optimal solutions are often acceptable, too. In this domain, more important requests are that solutions are found quickly and look realistic. In our framework, completeness means that the existence of an abstract solution (i.e., a sequence of abstract moves which can be mapped to a low-level solution) is guaranteed for any problem. We have not yet studied the completeness issue of our model in detail. In our Sokoban solver, we have chosen to trade completeness for efficiency. However, the lack of completeness does not seem to be a general property of the model. In simpler domains such as path-finding, we do not see any reason why the completeness should not be guaranteed.

The rest of the paper is structured as follows: In the next section we review the related work. In the third section we briefly introduce a formalization of our planning model. The next section describes the current status of our work and further research challenges that we plan to address in the future. The paper ends with our conclusion.

Related Work

Abstraction is a frequently used technique to reduce problem complexity in AI planning. Automatically abstracting planning domains has been explored by Knoblock (Knoblock 1994). His approach builds a hierarchy of abstractions by dropping literals from the problem definition at the previous abstraction level. Bacchus and Yang define a theoretical probabilistic framework to analyze the search complexity in hierarchical models (Bacchus & Yang 1994). They also use some concepts of that model to improve Knoblock's abstraction algorithm. In this work, the abstraction consists of problem relaxation. In our approach, abstraction means to reformulate a problem into an equivalent hierarchical representation. The abstract problem is solved independently from the initial problem formulation.

Long *et al.* use *generic types* and *active preconditions* to reformulate and abstract planning problems (Long, Fox, & Hamdi 2002). As a result of the reformulation, sub-problems of the initial problem are identified and solved by using specialized solvers. Our approach has similarities with this work. Both formalisms try to cope with domain-specific features at the local level, keeping the global problem as generic as possible. The difference is that we reformulate problems as a result of topological abstraction, whereas in the cited work reformulation is obtained by identifying various generic types of behavior and objects such as *mobile objects*.

Using topological abstraction to speed-up planning in a reinforcement learning framework has been proposed in (Precup, Sutton, & Singh 1997). In this work, the authors define macro actions as *offset-casual* policies. In such a policy, the probability of an atomic action depends not only on the current state, but also on the previous states and atomic actions of the policy. Learning macro actions in a grid robot planning domain induces a topological abstraction of the problem space.

Previous experiments showed that planning in a low-level Sokoban formulation was too hard for state-of-the-art generic planners (McDermott 1997; Junghanns & Schaeffer 1999). Culberson performed a theoretical analysis of Sokoban, showing that this domain is PSPACE-complete (Culberson 1997). The state-of-the-art Sokoban solvers are Junghanns' *Rolling Stone* (Junghanns 1999; Junghanns & Schaeffer 2001) and *deep green*, developed inside the Japanese Sokoban community (Junghanns 1999). These applications can find solutions for two thirds of the standard 90-problem test suite [1].

The LAP Formalism

LAP is a planning model based on a topological abstraction of the state representation. A clustering of the problem representation space is used to define boundaries between problem components. The clustering process aims to group together related atomic pieces and keep cluster interactions low. The abstraction allows us to decompose the initial problem into a hierarchy of sub-problems in a divide-and-conquer manner. For each cluster we define a local problem, which solves the local constraints of that cluster. The global problem uses an abstract problem description, where global states are characterized by states of abstract features. Each feature is a cluster that represents several atomic elements of the space.

At the global level, our abstraction approach leads to a much more compact state representation. For instance, a *room* in a robot planning domain is an abstract feature encoding many low-level objects such as atomic-size *squares*. Since one cluster is a complex feature representing several atomic features, cluster states can have many possible values. It is therefore natural to represent the global abstract states as tuples of cluster values. Using this representation, our abstraction model can be defined as a special case of the Simplified Action Structures (SAS) model (Bäckström & Klein 1991; Bäckström & Nebel 1995).

Next we formally define the LAP model as a special case of SAS structures. To do so, we first provide the SAS and SAS$^+$ definitions given in (Bäckström & Nebel 1995).

Definition 1 *A SAS structure* $\Phi = < \mathcal{M}, S, \mathcal{H} >$ *is defined by*

- *a finite set* $\mathcal{M} = \{i_1, ..., i_m\}$ *of state variable indices;*
- *a space of total states* $S = S_{i_1} \times S_{i_2} \times ... \times S_{i_k}$*, where*
  - S_j *is a domain of mutually exclusive values for the* j*th state component,*
  - $S_j^+ = S_j \cup \{u_j, k_j\}$ *is the extended domain, where* u_j *is the undefined value and* k_j *is the contradictory value for* S_j*;*

  $S^+ = S_{i_1}^+ \times S_{i_2}^+ \times ... \times S_{i_k}^+$ *is the space of partial states.*
- *a set* $\mathcal{H} = \{h_1, ..., h_n\}$ *of operators so that each* $h \in \mathcal{H}$ *has the structure* $h = < b(h), e(h), f(h) > \in S^+ \times S^+ \times S^+$ *where*
  - $b(h)$ *is the pre-condition of* h*,*
  - $e(h)$ *is the post-condition of* h*,*

[1]The test suite is available at `http://xsokoban.lcs.mit.edu/xsokoban.html`.

– *$f(h)$ is the prevail-condition of h (i.e., state components that do not change when h is applied).*

The undefined value is introduced to better express $b(h)$, $e(h)$, and $f(h)$. For instance, if an operator h leaves unchanged the jth component of a state, then $e(h)[j] = u_j$. Given a state $s \in S^+$, we define $dim(s) = \{i \in \mathcal{M} | s[i] \neq u_i\}$. The contradictory value is introduced for theoretical purposes only, so that a partial order is defined over S^+ and it becomes a lattice.

Definition 2 *A SAS$^+$-structure is a SAS-structure with the following additional constraints:*

- $\forall h \in \mathcal{H}, \forall j \in \mathcal{M}$: *$k_i$ does not appear in $b(h)$, $e(h)$, or $f(h)$;*
- $\forall h \in \mathcal{H} : \dim(b(h)) \subseteq \dim(e(h))$;
- $\forall h \in \mathcal{H} : \dim(e(h)) \cap \dim(f(h)) = \phi$;
- $\forall h \in \mathcal{H}, \forall i \in \mathcal{M} : b(h)[i] \neq u_i \Rightarrow b(h)[i] \neq e(h)[i]$

The first condition removes the contradictory value from the framework, so that it only has the theoretical meaning presented above. The second condition states that a component's value cannot be changed from a defined value to the undefined value. The third condition ensures that an operator cannot both change one component's value and yet preserve its value. The last condition states that components whose values do not change should be be part of $f(h)$.

In order to introduce LAP, we have to define the *unary* operators, the *binary* operators, and the *adjacency graph*. An operator $h \in \mathcal{H}$ is unary if $|dim(e(h))| = 1$. An operator $h \in \mathcal{H}$ is binary if $|dim(e(h))| = 2$. An unary operator changes only the value of one state component. We denote the set of unary operators by $\mathcal{H}^1$. A binary operator changes only the value of two state components. We denote the set of binary operators by $\mathcal{H}^2$. The *adjacency graph* is an undirected graph $\Gamma = (V(\Gamma), E(\Gamma))$, where $V(\Gamma) = \mathcal{M}$ and edges model the cluster inter-relationships.

Definition 3 *A LAP-structure $\Lambda =< \mathcal{M}, S, \mathcal{H} >$ is a SAS$^+$-structure that respects the following additional constraints:*

- $\mathcal{H} = \mathcal{H}^1 \cup \mathcal{H}^2$;
- $\forall h \in \mathcal{H} : \dim(b(h)) = \dim(e(h))$;
- *there is an adjacency graph $\Gamma = (V(\Gamma) = \mathcal{M}, E(\Gamma))$ defined for this model;*
- $\forall h \in \mathcal{H}^2 : dim(e(h)) \in E(\Gamma)$

The first condition states that all operators are either unary or binary. Equivalently, an abstract action h changes either one state component or two state components, leaving the rest of the tuple unchanged. In other words, the planning agent is only allowed to do local processing inside a cluster or perform an action affecting two adjacent clusters. This constraint is related to the unary-SAS, where an action is allowed to change only one component of a state. The second condition emphasizes that, when checking the preconditions of an operator, only the values of the components that are going to be changed matter. The graph Γ models cluster relationships. The last condition states that a transition between two clusters is possible only if the two clusters are neighbors.

Current Status and Future Work

At this stage of our work, we have achieved definite progress, which constitutes our basis for future research developments. We have introduced the LAP model, which represents our vision about using local abstraction to reduce the complexity of planning and heuristic search problems. We also applied the model to Sokoban (Botea, Müller, & Schaeffer 2002; Botea 2002; Botea, Müller, & Schaeffer 2003) and path-finding (Botea, Müller, & Schaeffer 2003), getting a clear indication about the potential of the approach.

There are several directions in which we plan to extend our current results. Since our understanding of the model is still in an early stage, we plan to explore it further, to better understand the strong and the weak points. We also intend to develop a formal description of the model and compare it to other abstraction models.

In the future we want to make steps toward building a unitary planning framework based on LAP. We want to develop domain-independent abstraction algorithms to be used as part of the model. The challenge is to take a non-abstracted problem and domain formulation written in PDDL and translate it to an equivalent abstract representation. One such technique should perform automatic clustering in the representation space. Also, it is interesting to study the trade-off between efficiency and using generic algorithms, especially for problem abstraction and solving local problems.

As part of an unified planning framework based on topological abstraction, the support of the planning language can be very important. The support for hierarchical planning should allow us to express an abstracted problem as a sum of sub-problems having different abstraction levels. The support for abstraction includes modeling the relation between low-level and abstract features, automatic abstraction, adaptive abstraction, and using hybrid state representations. Adaptive abstraction and hybrid state representation model the case when only a part of the problem space is abstracted. In such a situation, global states can be represented as a mixture of abstract features (for the abstracted problem components) and low-level features (for the part of the problem not abstracted yet). In addition, the abstraction can be performed gradually, as the planning agent discovers more about the problem topology. In (Botea, Müller, & Schaeffer 2003) we discuss in more detail how a more general version of PDDL could be useful to better model abstraction and hierarchical planning.

Another future work topic is to extend our work in applying LAP to domains such as Sokoban. In Sokoban we have solved 25 problems so far, as compared to about 60 for the state-of-the-art applications *Rolling Stone* and *deepgreen*. In this context, our goal is to surpass this performance, showing that hierarchical search can be more efficient than low-level search. The limitations of our system are in the lack of the domain-specific knowledge, not in the abstraction approach.

Conclusion

This paper presented a hierarchical planning approach based on topological abstraction. Topological abstraction is a powerful technique for reducing problem complexity in AI planning and single-agent search. The method is based on a clustering of the initial problem representation space. The clustering catches local relationships inside clusters and keep cluster interactions as limited as possible. In effect, the initial problem is decomposed into a two-level hierarchy of sub-problems, each being much simpler than the initial one. At the local level, each cluster has associated a local problem that solves the local constraints. There is also a global planning problem which uses clusters as features in the global state description. Since this model is useful in several application domains, it is worth to build a generic planning framework using topological clustering. As part of building this framework, we have shown what we have achieved so far and what are the challenges that we want to deal with in the future.

References

Bacchus, F., and Yang, Q. 1994. Downward Refinement and the Efficiency of Hierarchical Problem Solving. *Artificial Intelligence* 71(1):43–100.

Bacchus, F. 2001. AIPS'00 Planning Competition. *AI Magazine* 22(3):47–56.

Bäckström, C., and Klein, I. 1991. Planning in Polinomial Time: The SAS-PUBS Class. *Computational Intelligence* 7(3):181–197.

Bäckström, C., and Nebel, B. 1995. Complexity Results for SAS + Planning. *Computational Intelligence* 11(4):625–655.

Botea, A.; Müller, M.; and Schaeffer, J. 2002. Using Abstraction for Planning in Sokoban. To appear in *Proceedings of the 3rd International Conference on Computers and Games (CG'2002), Edmonton, Canada*.

Botea, A.; Müller, M.; and Schaeffer, J. 2003. Extending PDDL for Hierarchical Planning and Topological Abstraction. To appear in *Proceedings of the PDDL Workshop at the 13th International Conference on Automated Planning and Scheduling, Trento, Italy*.

Botea, A. 2002. Using Abstraction for Heuristic Search and Planning. In Koenig, S., and Holte, R., eds., *Proceedings of the 5th International Symposium on Abstraction, Reformulation, and Approximation*, volume 2371 of *Lecture Notes in Artificial Intelligence*, 326–327.

Culberson, J. 1997. SOKOBAN is PSPACE-complete. Technical report, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada. `ftp://ftp.cs.ualberta.ca/pub/TechReports/1997/TR97-02`.

Fox, M., and Long, D. 2002. The Third International Planning Competition: Temporal and Metric Planning. In *Preprints of The Sixth International Conference on AI Planning and Scheduling, Toulouse, France*, 115–118.

Junghanns, A., and Schaeffer, J. 1999. Domain-Dependent Single-Agent Search Enhancements. In *Proceedings IJCAI-99, Stockholm, Sweden*, 570–575.

Junghanns, A., and Schaeffer, J. 2001. Sokoban: Enhancing Single-Agent Search Using Domain Knowledge. *Artificial Intelligence* 129(1–2):219–251.

Junghanns, A. 1999. *Pushing the Limits: New Developments in Single-Agent Search*. Ph.D. Dissertation, University of Alberta.

Knoblock, C. A. 1994. Automatically Generating Abstractions for Planning. *Artificial Intelligence* 68(2):243–302.

Long, D.; Fox, M.; and Hamdi, M. 2002. Reformulation in Planning. In Koenig, S., and Holte, R., eds., *Proceedings of the 5th International Symposium on Abstraction, Reformulation, and Approximation*, volume 2371 of *Lecture Notes in Artificial Intelligence*, 18–32.

McDermott, D. 1997. Using Regression-Match Graphs to Control Search in Planning. `http://www.cs.yale.edu/HTML/YALE/CS/HyPlans/mcdermott.html`.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 21(2):35–55.

Precup, D.; Sutton, R.; and Singh, S. 1997. Planning with Closed-loop Macro Actions. In Working notes of the 1997 AAAI Fall Symposium on Model-directed Autonomous Systems, 1997.

Yap, P. 2002. Grid-based path-finding. In Cohen, R., and Spencer, B., eds., *Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence*, 44–55.