Downward Path Preserving State Space Abstractions (Extended Abstract)

Sandra Zilles
University of Regina
Department of Computer Science
Regina, Saskatchewan, Canada S4S 0A2
zilles@cs.uregina.ca

Robert C. Holte
University of Alberta
Department of Computing Science
Edmonton, Alberta, Canada T6G 2H8
holte@cs.ualberta.ca

Abstract

A problem that often arises in using abstraction is the generation of abstract states, called spurious states, that are—in the abstract space—reachable from some abstract image of a state s, but which have no corresponding state in the original space reachable from s. Spurious states can have a negative effect on pattern database sizes and heuristic quality.

We formally define a property—the downward path preserving property (DPP)—that guarantees an abstraction has no spurious states. Analyzing the computational complexity of *(i)* testing the DPP property for a given state space and abstraction and of *(ii)* determining whether this property is achievable at all for a given state space, results in strong hardness theorems. On the positive side, we identify formal conditions under which finding DPP abstractions is tractable.

Introduction

Abstraction is a popular technique for speeding up planning and search. The idea is to build a coarse "abstract" version of a given state space so that planning or search in the abstract space is rapid. This can be exploited in two ways: *(i)* by constructing a plan between two states in the original space by refining abstract plans found between the corresponding abstract states or *(ii)* by defining a heuristic function to guide planning and search by using actual distances in the abstract space as estimates of distances in the original space. A potential problem with abstraction is that it can introduce "spurious states". Given a state s in the original space, a spurious state is an abstract state that is reachable from the abstraction of s but is not the abstract image of any original state reachable from s.

The following example of 4 states $s_1, \ldots, s_4$ illustrates how spurious states can be generated. Assume s_2 is reachable from s_1 and s_4 is reachable from s_3 but neither s_3 nor s_4 are reachable from either s_1 or s_2. If an abstraction ψ identifies s_2 and s_3 with each other, but maps s_1 and s_4 to two separate abstract states, then the abstract state $\psi(s_4)$ is spurious with respect to s_1. It is reachable from $\psi(s_1)$ but has no pre-image in the original space that is reachable from s_1, see Figure 1.

$$\begin{tikzcd}
(a) & \boxed{s_1} \arrow[r] & \boxed{s_2} & \boxed{s_3} \arrow[r] & \boxed{s_4} \\
(b) & \boxed{\psi(s_1)} \arrow[r] & \boxed{\psi(s_2) = \psi(s_3)} \arrow[r] & \boxed{\psi(s_4)}
\end{tikzcd}$$

Figure 1: (a) Original state space and (b) abstract space with spurious state $\psi(s_4)$.

Spurious states can cause several difficulties. First, an abstract plan containing spurious states will be unrefinable, and if the length of the abstract plan is used as a heuristic it will often be overly optimistic because of "short-cuts" created in the abstract space by spurious states. Unrefinable abstract plans and low heuristic values can drastically slow down planning and search. Secondly, if abstract distances are stored in memory, as is done with pattern databases (PDBs), introduced in (Culberson and Schaeffer 1998), spurious states can increase the memory requirements dramatically.

In the heuristic search literature, the problem has been addressed by Holte and Hernádvölgyi (2004). They analyze different structural properties of state spaces and abstractions that are likely to cause spurious states. Recently Haslum et al. 2007—dealing with the problem of how to define "good" abstractions—report that PDB heuristics often turn out to be overly optimistic due to the existence of spurious states. Haslum, Bonet, and Geffner (2005) experimentally show that enforcing certain constraints when constructing the abstract space, and thus reducing the number of spurious states, can sometimes substantially speed up A*. However, there are no practical methods known so far to avoid spurious states completely; neither is there any formal research on the complexity of the problem of avoiding them.

This motivates a formal analysis of the problem of avoiding spurious states in abstraction. To this end we introduce what we call the "Downward Path Preserving" (DPP) property. This characteristic (*i.e.*, necessary and sufficient) property specifies state space abstractions that do not cause spurious states at all.

We study the computational complexity of two closely related problems, namely (A) to determine whether or not a given abstraction has the DPP property, and (B) to determine whether or not a given state space possesses a DPP abstraction at all.[1] Our complexity analyses show that both problems are, in general, hard to solve and that efficient general algorithms to produce DPP abstractions are thus unlikely to exist.

The negative results on the computational complexity are mitigated by tractability results for special cases. We identify simple formal conditions on state spaces that allow for finding DPP abstractions in polynomial time. Some typical problem domains turn out to be representable in a way that these conditions apply. However, some of them also have natural encodings that do not match the formal conditions of our theorems, and for which, moreover, it is not obvious that DPP abstractions exist at all. This shows that the way a problem domain is encoded is crucial for the design of DPP abstractions. Given two equivalent representations, one might immediately yield DPP abstractions, whereas the other might prevent DPP abstractions.

Formal definition of DPP abstractions

In this section we will formally define state spaces, abstractions, and the DPP property.

$|A|$ denotes the cardinality of a set A. If $\psi : A \to B$ is a mapping between sets A, B then $\psi(A) = \{\psi(a) \mid a \in A\} \subseteq B$. Let Σ, Γ be finite alphabets, $\Gamma \subseteq \Sigma$, $n \in \mathbb{N}$. $\psi : \Sigma^n \to \Gamma^n$ is called a string homomorphism if there is a mapping $\psi_0 : \Sigma \to \Gamma$ such that $\psi(\sigma_1, \dots, \sigma_n) = (\psi_0(\sigma_1), \dots, \psi_0(\sigma_n))$ for all $\sigma_1, \dots, \sigma_n \in \Sigma$.

Definition 1 *A state space is a triple $\mathcal{S} = (\Sigma, n, \Pi)$ where Σ is a finite alphabet, $n \in \mathbb{N}$, and $\Pi \subseteq \Sigma^n \times \Sigma^n$. Every $s \in \Sigma^n$ is called a state and every pair $(s, s') \in \Pi$ is called an edge from state s to state s'.*

Definition 2 *Let $\mathcal{S} = (\Sigma, n, \Pi)$ be a state space, $s, s' \in \Sigma^n$. s' is reachable from s (in $\mathcal{S}$) if there is a sequence $\pi = (s_1, s_2, \dots, s_z) \in (\Sigma^n)^+$ such that $s_1 = s$, $s_z = s'$, and $(s_i, s_{i+1}) \in \Pi$ for all $i \in \{1, \dots, z-1\}$. Define $\Delta(s, \mathcal{S}) = \{s' \in \Sigma^n \mid s'$ is reachable from s in $\mathcal{S}\}$.*

We assume that the set of all states in a state space $\mathcal{S}$ is always the full set of n-tuples over Σ. In contrast to that it is also common to define a state space via a seed state and all states reachable from that state (edges and thus reachability might be defined in the form of operators). In such a definition, the set of states in $\mathcal{S}$ would only be one connected component of Σ^n. These different perspectives on state spaces do not play a role in this paper and so for simplicity we always assume the set of states to be equal to Σ^n.

We follow the PSVN notation introduced in (Hernádvölgyi and Holte 1999) for state space representation. Here we define edges via parameterized operators, so that one operator can represent a large set of edges.

For instance, let $\Sigma = \{a, b, c\}$, $n = 4$ and $X = \{x_1, x_2\}$ a set of variables. The operator $\langle x_1, c, x_1, x_2 \rangle \to \langle b, x_1, x_2, x_2 \rangle$ means that every state s which has a 'c' in component 2, identical entries σ in components 1 and 3, and any value σ' in component 4 has an outgoing edge to the state that has a 'b' in component 1, σ in component 2, and σ' in components 3 and 4. For example, $(\langle a, c, a, a \rangle, \langle b, a, a, a \rangle)$ and $(\langle b, c, b, a \rangle, \langle b, b, a, a \rangle)$ are edges induced by this operator, whereas neither $(\langle a, a, a, a \rangle, \langle b, a, a, a \rangle)$ nor $(\langle a, c, a, a \rangle, \langle b, b, a, a \rangle)$ are. This notation is formalized in Definition 3.

Definition 3 *Let $\mathcal{S} = (\Sigma, n, \Pi)$ be a state space and $X = \{x_i \mid i \in \mathbb{N}\}$ a set of variables, $X \cap \Sigma = \emptyset$. Any n-tuple $p = \langle y_1, \dots, y_n \rangle \in (\Sigma \cup X)^n$ is called a state pattern.*
Let $p = \langle y_1, \dots, y_n \rangle$ and $p' = \langle y'_1, \dots, y'_n \rangle$ be state patterns. $p \to p'$ is called an operator if, for all $i \in \{1, \dots, n\}$, $y'_i \in \Sigma$ or $y'_i = y_j$ for some $j \in \{1, \dots, n\}$. A state $s = \langle \sigma_1, \dots, \sigma_n \rangle \in \Sigma^n$ matches p if for all $i \in \{1, \dots, n\}$ the following conditions hold.

(1) $y_i = \sigma_i$ or $y_i \in X$,
(2) if $y_i = y_j$ for some $j \in \{1, \dots, n\}$ then $\sigma_i = \sigma_j$.

An edge (s, s') matches the operator $o = p \to p'$ if s matches p, s' matches p', and for all $i, j \in \{1, \dots, n\}$, $y'_i = y_j \Rightarrow s'_i = s_j$.
(Σ, n, O) is a representation for $\mathcal{S}$ if O is a set of operators such that $(s, s') \in \Pi$ if and only if there is an $o \in O$ such that (s, s') matches o.

An abstraction of a state space $\mathcal{S}$ is a state space $\mathcal{S}_\psi$ with a mapping ψ from the states in $\mathcal{S}$ to the states in $\mathcal{S}_\psi$. The edges are considered to be induced by ψ.

Definition 4 *Let $\mathcal{S} = (\Sigma, n, \Pi)$ be a state space, $\Gamma \subseteq \Sigma$, and $m \in \mathbb{N}$. Any mapping $\psi : \Sigma^n \to \Gamma^m$ induces a state space $\mathcal{S}_\psi = (\Gamma, m, \Pi_\psi)$ over Γ^m where, for all $t, t' \in \Gamma^m$ we have $(t, t') \in \Pi_\psi$ if and only if there are states $s, s' \in \Sigma^n$ such that $\psi(s) = t$, $\psi(s') = t'$, and $(s, s') \in \Pi$. $\mathcal{S}_\psi$ is called an abstraction of $\mathcal{S}$, ψ is called an abstraction mapping.*

[1]We focus only on the two most common types of abstractions, namely projection and domain abstraction.

Thus ψ is a graph homomorphism between (Σ^n, Π) and (Γ^m, Π_ψ), where without loss of generality and just for simplicity we assume $\Gamma \subseteq \Sigma$. Note that our definition precludes the abstract space containing edges that are not induced by ψ (i.e., Π_ψ is the minimal set of edges for which ψ is a graph homomorphism between (Σ^n, Π) and (Γ^m, Π_ψ)).[2] So all hardness results that we prove below for this special case also hold in the general case where Π_ψ is only a subset of the edges in the abstract state space.

We consider two types of abstraction. In domain abstraction, some of the alphabet symbols are identified with each other; in projection, some of the state variables are ignored.

> *Domain abstraction:* $m = n$ but $|\Gamma| < |\Sigma|$. A domain abstraction[3] mapping is a surjective string homomorphism $\psi : \Sigma^n \to \Gamma^n$ with $\psi(\psi(\sigma)) = \psi(\sigma)$ for all $\sigma \in \Sigma$. The trivial case is $|\Gamma| = 1$.
>
> *Projection:* $\Gamma = \Sigma$ but $m < n$. A projection mapping ψ is defined via a subset $\{i_1, \ldots, i_m\} \subset \{1, \ldots, n\}$ such that $\psi(\sigma_1, \ldots, \sigma_n) = (\sigma_{i_1}, \ldots, \sigma_{i_m})$ for all $\sigma_1, \ldots, \sigma_n \in \Sigma$. The trivial case is $m = 0$.

If $\mathcal{S} = (\Sigma, n, \Pi)$ is a state space and ψ an abstraction mapping then $\psi(\Delta(s, \mathcal{S})) \subseteq \Delta(\psi(s), \mathcal{S}_\psi)$ holds for every $s \in \mathcal{S}$ by definition. The opposite inclusion does not necessarily hold—and this is what causes spurious states. For any given original state s^* (not necessarily the start or goal state), a spurious state with respect to s^* is an abstract state $t \in \Delta(\psi(s^*), \mathcal{S}_\psi) \setminus \psi(\Delta(s^*, \mathcal{S}))$, *i.e.*, an abstract state that is reachable from the abstract image of s^* but has no pre-image in the original state that is reachable from s^*. For any fixed state s^*, an abstraction ψ avoiding unwanted spurious states would fulfill $\Delta(\psi(s^*), \mathcal{S}_\psi) = \psi(\Delta(s^*, \mathcal{S}))$.

Definition 5 *Let $\mathcal{S} = (\Sigma, n, \Pi)$ be a state space and ψ an abstraction mapping. ψ is called a downward path preserving (DPP) abstraction of $\mathcal{S}$ if $\psi(\Delta(s, \mathcal{S})) = \Delta(\psi(s), \mathcal{S}_\psi)$ for all $s \in \Sigma^n$.*

DPP abstractions are hard to find

We assume that abstraction mappings can be computed in polynomial time, so that they are never the bottleneck when we obtain hardness results. We analyze the following decision problems.

IS-DPP. *Given a state space $\mathcal{S}$ and an abstraction mapping ψ (of type domain abstraction or projection), decide whether or not ψ is a DPP abstraction of $\mathcal{S}$.*

EXIST-DPP. *Given a type of abstraction (domain abstraction or projection), a state space $\mathcal{S}$ over Σ^n, $\Gamma \subseteq \Sigma$ and a number m (where either $n = m$ (domain abstraction) or $\Sigma = \Gamma$ (projection)), decide whether or not there is a non-trivial DPP abstraction mapping ψ of $\mathcal{S}$, of the given type, inducing a state space $\mathcal{S}_\psi$ over Γ^m.*

Hardness results on the problem of deciding, given a state space and two different states s and s', whether s is reachable from s', cf. (Bäckström 1992), help proving the hardness of the IS-DPP problem. However none of the proofs of the following hardness results are trivial; the reader is referred to (Zilles, Ball, and Holte 2009).

Theorem 1 *(1) IS-DPP is PSPACE-hard for either type of abstraction.*
(2) IS-DPP is in P for either type of abstraction if the dimension n is fixed a priori.
(3) EXIST-DPP is PSPACE-hard for either type of abstraction.
(4) EXIST-DPP is in P for projection if the dimension n is fixed a priori.
(5) EXIST-DPP is NP-hard for domain abstraction even if the dimension n is fixed with $n > 2$ a priori.

When DPP abstractions are easy to find

From a practical point of view, easily testable criteria that help us to design DPP abstractions for given state spaces are very interesting. This section provides such conditions for projection (Definition 6 and Theorem 2) and for domain abstraction (Definition 7 and Theorem 3). In particular they allow for finding DPP abstractions in polynomial time.

Definition 6 *Let (Σ, n, O) be a representation for a state space. Let $o = \langle y_1, \ldots, y_n \rangle \to \langle y'_1, \ldots, y'_n \rangle$ be an operator in O and $I \subseteq \{1, \ldots, n\}$. We say that o is closed on I if the following three conditions are satisfied.*

(1) $y_i \notin \Sigma$ for all $i \in I$.
(2) $y_i \neq y_j$ for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$.
(3) $\{y'_i \mid i \in I\} \subseteq \{y_i \mid i \in I\}$.

For example, in the Rubik's Cube domain (Korf 1997), every operator is closed on the set of indices of variables encoding the corner cubies. That means the effects on the corner cubie variables are given by shuffling and/or copying the corner cubie variables in the operator's preconditions (the latter not containing constants or duplicates of variable names). The same holds for the edge cubies. Our formal criterion allowing for DPP projections reads as follows.

[2]This equals the notion of homomorphism for abstraction mappings in (Helmert, Haslum, and Hoffmann 2007).

[3]Here the term "domain" refers to the domain Σ of the variables used for state space encodings, while often we use the term "domain" to refer to the general problem domain, *e.g.*, the Blocks World domain with k blocks.

Theorem 2 *Let (Σ, n, O) be a representation for a state space $\mathcal{S}$. Suppose there is a disjoint partitioning $I_1 \cup \ldots \cup I_z = \{1, \ldots, n\}$, such that o is closed on I_j for every $o \in O$ and every $j \in \{1, \ldots, z\}$. Let $j \in \{1, \ldots, z\}$ and $I_j = \{j_1, \ldots, j_l\}$. Then the mapping ψ with $\psi(\sigma_1, \ldots, \sigma_n) = (\sigma_{j_1}, \ldots, \sigma_{j_l})$ defines a DPP abstraction of $\mathcal{S}$.*

Every variable in the standard representation of the Rubik's Cube domain, cf. (Korf 1997), encodes either a property of a corner cubie or a property of an edge cubie. Since every operator is both closed on the set of indices of corner cubie variables and closed on the set of indices of edge cubie variables, we can project out either all corner cubie variables or all edge cubie variables without violating the DPP property.

Definition 7 *Let (Σ, n, O) be a representation for a state space. Let $o = \langle y_1, \ldots, y_n \rangle \rightarrow \langle y'_1, \ldots, y'_n \rangle$ be an operator in O and $\Sigma_0 \subseteq \Sigma$. We say that o is independent on Σ_0 if the following two conditions are satisfied.*

(1) $y_i \notin \Sigma_0$ for all $i \in \{1, \ldots, n\}$.
(2) $y_i \neq y_j$ for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$.

For instance, in the standard representation of the Sliding Tile puzzle, every operator is independent on the set of the names of tiles. The only constant symbol ever appearing in any operator is the symbol B encoding the blank; the symbols encoding names of tiles never occur in the operators. Theorem 3 then says that every domain abstraction that shrinks the set of tile names and leaves the blank symbol untouched will be DPP. It applies if we set $\Sigma_1 = \{B\}$ and let Σ_0 be the set of symbols representing tile names.

Theorem 3 *Let $(\Sigma_0 \cup \Sigma_1, n, O)$ be a representation for a state space $\mathcal{S}$, where $\Sigma_0 \cap \Sigma_1 = \emptyset$. Suppose every $o \in O$ is independent on Σ_0. Then every domain abstraction mapping ψ with $\psi(\Sigma_0) \subseteq \Sigma_0$ and $\psi(\sigma) = \sigma$ for every $\sigma \in \Sigma_1$ defines a DPP abstraction of $\mathcal{S}$.*

As Theorems 2 and 3 show, certain properties of state spaces immediately allow for finding DPP abstractions in polynomial time. Since these properties concern not the problem domain as such but its encoding as a state space, there might be two encodings defining isomorphic state spaces for the same problem domain such that one of them has DPP projections (or DPP domain abstractions) while the other does not.

Probably not every planning or search domain can be encoded in a way that DPP abstractions can easily be designed. Moreover, even if this is possible for a certain problem domain, it is far from trivial to (automatically) find such encodings. Nevertheless there is a rich history of research on automatic problem reformulation that addresses this issue, e.g., (Amarel 1968; Benjamin 1989; Korf 1980; Lowry 1988; Van Baalen 1992).

Some examples of planning domains, including Blocks World, show that, if one deviates from standard STRIPS encodings, quite intuitive encodings that match Theorems 2 and 3 can be found. Hence a careful design of problem domain encodings may make the design of DPP abstractions and thus maybe also planning and search easier, cf. (Zilles, Ball, and Holte 2009).

Acknowledgments

This work was supported by NSERC and the Alberta Ingenuity Fund.

References

Amarel, S. 1968. On representations of problems of reasoning about actions. In Michie, D., ed., *Machine Intelligence*, volume 3. 131–171.

Bäckström, C. 1992. Equivalence and tractability results for SAS$^+$ planning. In *KR 1992*, 126–137.

Benjamin, D. P. 1989. *Change of Representation and Inductive Bias.* Kluwer Academic Publishers.

Culberson, J., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, 1007–1012.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *AAAI*, 1163–1168.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Hernádvölgyi, I., and Holte, R. 1999. PSVN: A vector representation for production systems. Technical Report TR-99-04, Univ. Ottawa Computer Science.

Holte, R., and Hernádvölgyi, I. 2004. Steps towards the automatic creation of search heuristics. Technical Report TR04-02, Dept. of Comp. Sci., Univ. Alberta.

Korf, R. E. 1980. Towards a model of representation changes. *Artificial Intelligence* 14(1):41–78.

Korf, R. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI*, 700–705.

Lowry, M. R. 1988. Invariant logic: A calculus for problem reformulation. In *AAAI*, 14–18.

Van Baalen, J. 1992. Automated design of specialized representations. *Artificial Intelligence* 54(1):121–198.

Zilles, S.; Ball, M.; and Holte, R. 2009. Downward path preserving state space abstractions. Technical Report TR09-04, Dept. of Comp. Sci., Univ. Alberta.