Using Coarse State Space Abstractions to Detect Mutex Pairs

Mehdi Sadeqi
Computer Science Department
University of Regina
Regina, SK, Canada S4S 0A2
(sadeqi2m@cs.uregina.ca)

Robert C. Holte
Computing Science Department
University of Alberta
Edmonton, AB, Canada T6G 2E8
(holte@cs.ualberta.ca)

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

Abstract

A mutex pair in a state space is a pair of assignments of values to state variables that does not occur in any reachable state. Detecting mutex pairs is a problem that has been addressed frequently in the planning literature. In this paper, we present the Coarse Abstraction (CA) method, a new efficient method for detecting mutex pairs in state spaces represented with multi-valued variables. CA detects mutex pairs based on exhaustive search in a collection of very small abstract state spaces. While in general CA may miss some mutex pairs, we provide a formal guarantee that CA finds all mutex pairs under a simple and quite natural condition. Using this formal guarantee, we prove that these properties hold for a range of common benchmark domains. We also show that CA can find all mutex pairs even if the formal guarantee is not satisfied. Finally, we show that CA's effectiveness depends on how the domain is represented, and that it can fail to find mutex pairs in some domains and representations.

Introduction

In the context of planning or search in a given state space, the term *mutually exclusive pair* (of facts), or *mutex pair* for short, refers to a pair of variable-value assignments that do not co-occur in any state that is reachable from the start state. In the case of a binary state space representation, such a pair corresponds to a pair of grounded atoms,[1] but in this paper we will assume multi-valued domain representations. For instance, in a Blocks World domain, in which named blocks can be stacked onto each other, an example of a mutex pair would be a pair of grounded atoms (or of variable-value assignments) that represents the facts *Block a is on top of Block b* and *Block b is on top of Block a*.

Mutex pair detection has long been in use for improving the performance of planning systems, in particular for the purpose of search space pruning. In regression planning, nodes in the search space correspond to partial assignments of state variables, and edges correspond to actions. Search proceeds backward from the goal until reaching a node whose assignment is true in the start state. One way of making regression planning more efficient is to prune nodes that are "impossible" in the sense that they contain mutex pairs. The original success of mutex detection in Graphplan (Blum and Furst 1995) led to the development of many efficient planners incorporating mutex detection, such as temporal planners, different SAT-based planners and planners based on constraint programming. SATPLAN04 (Kautz 2004), BLACKBOX (Kautz and Selman 1996), DISCOPLAN (Gerevini and Schubert 2000), MIPS (Edelkamp and Helmert 2001) and FD (Helmert 2006) are some examples of planners that successfully integrated mutex constraints in their reasoning process (see the section Related Work for more details).

Mutex detection can also be applied for improving the quality of heuristic functions derived from abstractions. Heuristic functions estimate, for any state s, the distance from s to a goal state. Heuristic search algorithms like A* and IDA* are guaranteed to find optimal solutions when using *admissible heuristics*, i.e., heuristic functions that never overestimate the true distances. One popular method for obtaining admissible heuristics is to create an abstract version of the original state space and to use the true distances in the abstract state space as heuristic values. The key to the efficiency of A* and IDA* is the quality of the heuristic values: the closer the heuristic values are to the true distances, the more effective they will be in speeding up search. Unfortunately, standard efficient methods known for enumerating abstract state spaces may include abstract states to which no reachable original state is mapped by the abstraction. Such abstract states are called spurious; they may create short-cuts in the abstract space and thus lower heuristic values (Zilles and Holte 2010). In many cases, spurious states contain mutex pairs. Hence, by removing some of the shortcuts created by spurious abstract states, mutex detection can help to improve the quality of a heuristic resulting from an abstraction, and thus to speed up search.

We call a mutex detection method "complete" if it finds all mutex pairs. Existing mutex pair detection algorithms usually make a compromise in the number of detected mutex pairs for the computational complexity of the algorithm. Various methods differ in the number and type of mutex pairs they detect (see Related Work for more details). To the best of our knowledge, there are no formal conditions discussed in the literature under which any of the existing

[1] We treat propositional logic variables, such as those commonly used in planning, as variables that can take on one of two values (true and false).

mutex detection methods are guaranteed to be complete.

In this paper, we propose the Coarse Abstraction method (CA), a new, efficient method that uses very coarse abstract versions of the original state space to find mutex pairs. CA is guaranteed to be complete under certain natural, easy-to-test conditions on the state space representation. We demonstrate that CA is complete for specific representations of many of the typical benchmark domains (namely permutation problems, such as Scanalyzer, and non-trivial Sliding-Tile Puzzles). Empirically, we also verify that CA is very efficient on the Scanalyzer and Sliding-Tile Puzzle domains. It is important to emphasize two remarkable properties of the formal condition under which CA's completeness is guaranteed: (i) it is very natural—we expect it to apply to many domain representations, and (ii) the literature offers efficient methods for verifying the formal condition by simple syntactic checks of the operators in the domain representation (Zilles and Holte 2010). We further prove that CA can be complete even in cases when the conditions guaranteeing its completeness do not apply, for example in the case of a particular representation of the Blocks World with distinct table positions.

In a small experiment on the Towers of Hanoi, on a different representation of the Blocks World, and on a constrained version of the Sliding-Tile Puzzle, CA performed poorly. It is interesting that CA is provably complete on one representation of the Blocks World, while it finds none of the mutex pairs in another representation of the same domain. This shows that the success of CA depends on both the domain and on how the domain is represented. We conclude that CA is not a general-purpose mutex detection method, but that it clearly outperforms existing methods in certain domain representations.

Most mutex-detection methods apply only to binary-valued domain representations, not to multi-valued ones (an exception is a recent study by Solèr (2012)). CA is the opposite, it is applicable only to multi-valued representations. This is a minor limitation since many planners convert their given binary representation into a multi-valued representation in a preprocessing step.

Coarse Abstraction Mutex Detection

Assume that states are represented as variable-value pairs in m variables. We denote the variables with $V_1, \ldots, V_m$, so that the state vector $\langle v_1, \ldots, v_m \rangle$ corresponds to the assignment vector $(V_1 = v_1, \ldots, V_m = v_m)$. Sometimes we also use V, W, X, Y to denote state variables. Propositional logic variables, such as those commonly used in planning, are treated as variables that can take on one of two values (true and false).

With this convention, we define the notions of reachable state and mutex pair formally.

Definition 1 *Let s^* be any fixed state. Let Σ be the value set of the state variables.*

Since s^* is fixed, we will simply call a state reachable if it is reachable from s^*. For any i, j with $1 \leq i < j \leq m$ and any $v_i, v_j \in \Sigma$, the partial original state $(V_i = v_i, V_j = v_j)$ is a reachable pair if there are v_k, for $k \in \{1, \ldots, m\} \backslash \{i, j\}$ such that $\langle v_1, \ldots, v_m \rangle$ is a reachable state; otherwise $(V_i = v_i, V_j = v_j)$ is a mutex pair.

Our mutex detection method builds on the notion of abstraction; in particular it uses domain abstractions.

Definition 2 *An* abstraction *mapping Ψ maps any state s in the original state space to an abstract state $\Psi(s)$. A* domain abstraction *Ψ_f is induced by a mapping f on Σ in the following way: if f maps values v to values $f(v)$, and $s = \langle v_1, \ldots, v_m \rangle$ then $\Psi_f(s) = \langle f(v_1), \ldots, f(v_m) \rangle$.*

For example, if $s = \langle 1, 2, 2, 0 \rangle$, $f(1) = f(2) = dc$, and $f(0) = 0$, then $\Psi_f(s) = \langle dc, dc, dc, 0 \rangle$. Here dc can be thought of as a "don't care" value, i.e., the domain abstraction Ψ_f keeps the value 0 unchanged and maps the values 1 and 2 to "don't care".

As is standard practice in planning and heuristic search, we assume that edges between abstract states are generated by applying the abstraction mapping to the operators.

The Coarse Abstraction (CA) method for identifying mutex pairs builds on the following fact: if V and W are any two state variables and there is a reachable state with variable assignments $V = v$ and $W = w$, then *any* abstraction Ψ that does not project out V and W will create a reachable abstract state with $V = \Psi(v)$ and $W = \Psi(w)$. The set of pairs $(V = v, W = w)$ that are reachable in the original space is therefore a subset of the pre-image[2] of the set of pairs $(V = \Psi(v), W = \Psi(w))$ reachable in the abstract space Ψ. This means that any pair $(V = v, W = w)$ in the original space that is *not* in this pre-image is certainly a mutex pair.

CA works as follows. For each pair (v, w) of constants in Σ (including pairs of the form (v, v)):

1. define the domain abstraction $\Psi_{v,w}$ (called a *coarse abstraction*) that leaves v and w distinct (or only v for pairs of the form (v, v)) and maps all other constants to the same constant ("don't care"),
2. enumerate the abstract states that are reachable from $\Psi_{v,w}(s^*)$ (using exhaustive search from $\Psi_{v,w}(s^*)$),
3. for each pair (X, Y) of state variables such that $(X = v, Y = w)$ occurs in a state thus enumerated mark $(X = v, Y = w)$ as "non-mutex".
4. mark each remaining pair $(X = x, Y = y)$ as "mutex".

A Guarantee on CA

While any pair considered mutex by CA is indeed mutex, CA is in general not guaranteed to find all mutex pairs. This section is concerned with finding conditions on the domain representation that (i) guarantee that CA will find all mutex pairs and (ii) can be easily checked, e.g., by a syntactic check of the operators that can be computed efficiently.

One such guarantee can be obtained under a simple and quite natural condition, based on the notion of *spurious state*. A spurious state is an abstract state that is reachable

[2]The pre-image of the set of pairs $(V = \Psi(v), W = \Psi(w))$ is the set of all pairs in the original state space that are mapped to this set by the abstraction.

from the abstract version of s^*, while it has no reachable pre-image under the abstraction mapping (Zilles and Holte 2010).

Definition 3 *Let Ψ be any abstraction mapping. An abstract state a is called* spurious *if a is reachable from $\Psi(s^*)$, but there is no original state t such that $\Psi(t) = a$ and t is reachable from s^*.*

It turns out that the absence of spurious states in the coarse abstractions used by CA is a sufficient condition for CA to be complete.

Theorem 1 *If all the coarse abstractions are free of spurious states, then CA finds all mutex pairs.*

Proof Suppose CA considers the variable assignment $(V = v, W = w)$ non-mutex. This means that $(V = v, W = w)$ is marked non-mutex through the coarse abstraction $\Psi_{v,w}$. Since $\Psi_{v,w}$ generates no spurious states, there is a reachable state in the original state space that has the assignment $(V = v, W = w)$. Hence this pair is non-mutex. Consequently, no mutex pair is mistakenly flagged "non-mutex". ■

Unfortunately, deciding whether or not an abstraction contains spurious states is PSPACE-complete in general (Zilles and Holte 2010). In special cases though this decision problem can be solved efficiently via simple syntactic checks on the operators. Zilles and Holte (2010) provide a number of such conditions, each of which guarantees an abstraction of a particular type (domain abstraction or projection) to be free of spurious states. The following is a special case of one of their general conditions.

Theorem 2 *(Zilles and Holte 2010) Suppose no original operator has any preconditions. Then any domain abstraction is free of spurious states.*

In other words, if every operator applies to every state, then any domain abstraction is free of spurious states. This theorem applies to natural representations of any state space that is a finite mathematical group, such as Rubik's Cube, the pancake puzzle, TopSpin, and Scanalyzer (Helmert and Lasinger 2010). CA is therefore complete on all such domains. In the rest of the paper we use Scanalyzer as a representative of such state spaces.

We also show, using domain-specific knowledge, that the coarse abstractions of two common representations of non-trivial versions of the Sliding-Tile Puzzle satisfy the conditions of Theorem 1, thereby proving that CA is complete on this domain as well.

Application Case 1: Scanalyzer

In the n-Belt Scanalyzer domain, a state describes the placement of n plant batches on n conveyor belts along with information indicating which batches have been "analyzed". (For a detailed description of this domain, see (Helmert and Lasinger 2010).) In a *rotate* move, a batch can be switched from one conveyor belt in the upper half (A, B and C in Figure 1) to one in the lower half (D, E and F in Figure 1) and vice versa. In a *rotate-and-analyze* move, a batch can simultaneously be transferred and analyzed from the topmost conveyor belt to the bottommost one while the batch at the bottommost conveyor belt is moved to the topmost one without any change to its "analyze" state. Once a batch is analyzed, it will remain analyzed.

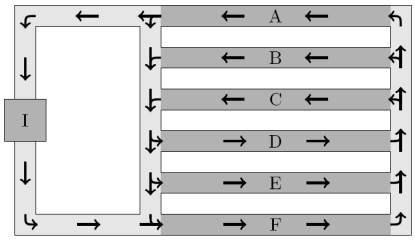

Figure 1: 6-Belt Scanalyzer, from (Helmert and Lasinger 2010). Arrows indicate legal moves.

We use a representation of the n-Belt Scanalyzer in production system vector notation (PSVN) (Hernádvölgyi and Holte 1999). Here, a state is encoded as a vector of length $2n$ in which each belt corresponds to two components: the name of the batch on that belt and a flag indicating whether that batch is analyzed. This representation uses a set of operators each of which applies to every state in the domain.

The state s^*, again the standard goal state for this domain, corresponds to having all plant batches analyzed and replaced back on their original conveyor belts.

We can prove that with the above PSVN representation of Scanalyzer, CA will detect all mutex pairs, independent of the size of the domain.

Theorem 3 *CA finds all mutex pairs in the Scanalyzer domain of any size, using the representation described above.*

We will show that the coarse abstractions are free of spurious states and thus obtain Theorem 3 immediately from Theorem 1.

Since no operator in our encoding of the Scanalyzer domain has any preconditions, Theorem 2 immediately proves the following lemma.

Lemma 1 *No coarse abstraction of the Scanalyzer domain of any size, using the representation described above, contains any spurious states.*

Together with Theorem 1 we obtain that CA is complete on our representation of the Scanalyzer domain, i.e., Theorem 3 is proven. The same proof applies to any state space representation in which the operators have no preconditions.

Application Case 2: Sliding-Tile Puzzle

The $n \times \ell$-Sliding-Tile Puzzle represents an $n \times \ell$ grid, in which tiles numbered 1 through $n \cdot \ell - 1$ each fill one grid position and the remaining grid position is blank. A move consists of swapping the blank with an adjacent tile. Figure 2 shows a state of the 3×3-puzzle, often also called 8-puzzle, where the blank is depicted with a black grid position.

We use two representations of this domain.

In the *standard representation*, states are represented as vectors of length $n \cdot \ell$, where each component corresponds to

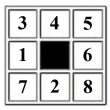

Figure 2: A state of the 3×3-puzzle.

a grid position and contains a value in $\{1, 2, \ldots, n \cdot \ell - 1, B\}$, representing the number of the tile in this position (B, if the position is blank). For example, the state in Figure 2 would be represented by $\langle 3, 4, 5, 1, B, 6, 7, 2, 8\rangle$. In the *dual representation* of the $n \times \ell$-puzzle, a vector component corresponds to either the blank or one of the tiles. The value of a vector component is an integer in $\{1, \ldots, n \cdot \ell\}$, representing the grid position at which the corresponding tile is located. For example, the state in Figure 2 would be encoded as $\langle 4, 8, 1, 2, 3, 6, 7, 9, 5\rangle$, where the i^{th} component, for $i \leq 8$, holds the position of tile i, and the 9^{th} component holds the position of the blank.

As the state s^*, we choose a goal state for this domain, which contains the blank in the bottom right corner of the grid, while the remaining grid positions contain tiles with increasing numbers, row by row from top to bottom, each row being filled from left to right. However, all our results on this domain hold true independent of the choice of s^*.

We will now prove that CA is complete on both representations of the Sliding-Tile Puzzle.

Theorem 4 *CA finds all mutex pairs in any Sliding-Tile Puzzle domain of size $n \times \ell$ for $n, \ell \geq 2$ and $n \times \ell \geq 5$, represented in standard representation or in dual representation.*

The proof consists of the following two lemmas, which state that, for either case of domain representation, the coarse abstractions are free of spurious states. Theorem 4 then is an immediate consequence of Theorem 1. The proofs of the two lemmas will make use of the fact that swapping the locations of two numbered tiles in a state of a Sliding-Tile Puzzle domain will always change the "parity" of a state, mapping a reachable state to an unreachable one and vice versa (Johnson and Story 1879).

Lemma 2 *No coarse abstraction of any Sliding-Tile Puzzle domain of size $n \times \ell$ for $n, \ell \geq 2$ and $n \times \ell \geq 5$, given in standard representation, contains any spurious states.*

Proof Let $\Psi_{i,j}$ (with $i, j \in \{blank, 1, ..., (n \times \ell - 1)\}$) be a coarse abstraction of a Sliding-Tile Puzzle domain of size $n \times \ell$ for $n, \ell \geq 2$ and $n \times \ell \geq 5$, given in standard representation, and let t be a resulting abstract state. Let i and j be the symbols kept by the coarse abstraction $\Psi_{i,j}$, i.e., the symbols not mapped to "don't care".

Let s be any original state in the pre-image of t. Consider the state s' that is created from s by swapping the locations of two distinct numbered tiles named i' and j' chosen from the set of tile names that $\Psi_{i,j}$ maps to "don't care", i.e., any two numbered tiles whose names are not in $\{i, j\}$. Since there are at least 4 tiles when $n, \ell \geq 2$ and $n \times \ell \geq 5$, two such tiles must exist. Since the names i' and j' of the swapped tiles are both mapped to "don't care" by $\Psi_{i,j}$, the state s' is in the pre-image of t as well. Consequently, the pre-image of t contains two states s and s' that can be obtained from each other by swapping two tiles. One of these states must be reachable. This implies that t is not spurious. ■

Lemma 3 *No coarse abstraction of any Sliding-Tile Puzzle domain of size $n \times \ell$ for $n, \ell \geq 2$ and $n \times \ell \geq 5$, given in dual representation, contains any spurious states.*

Proof The proof is similar to that of Lemma 2—one simply replaces all references to tile names by location names. ■

It should be noted that Lemmas 2 and 3 do not transfer to the case of the 2×2-Sliding-Tile Puzzle or any $n \times 1$-Sliding-Tile Puzzle for $n > 2$. In either of these cases, for the standard representation, any coarse abstraction that maps at least one tile and the blank to "don't care" will allow one of the tiles whose name is kept distinct to move to a position that it cannot assume in any reachable state. For example, in the 3×1 version of the puzzle, if both the blank and tile 2 are mapped to the same name, then tile 1 can assume any position on the grid, while one position is impossible for tile 1 in any reachable state. A similar argument applies to the dual representation.

Note also that our verification of CA's completeness on the Sliding-Tile Puzzle is not as straightforward as for the Scanalyzer and requires domain-specific knowledge. However, we believe that the generality of the condition in Theorem 1 (and of the conditions guaranteeing there are no spurious states, derived by Zilles and Holte (2010)) will allow a formal guarantee of completeness of CA to be widely applicable to cases in which it can be tested efficiently and without domain-specific knowledge.

Other Cases in Which CA Is Complete

The condition in Theorem 1 is not necessary—it can happen that some coarse abstractions generate spurious states and yet CA is complete. An example of this phenomenon is obtained from a particular representation of the Blocks World with distinct table positions.

Application Case 3: Blocks World with Distinct Table Positions

In the n-Blocks World with p distinct table positions, a state is a constellation of n blocks stacked on a table with p named (distinct) positions, where up to one block can be located in a "hand". In every move, either the empty hand picks up the top block off a stack of blocks (including a stack of size 1) or the hand places the block it is holding onto an empty table position or on top of a stack of blocks.

In one representation of this domain, called the *top representation*, a state vector has $1+p+n$ components, each containing either the value 0 or one of n possible block names $(1, ..., n)$: (i) the value of the first component is the name of the block in the hand or 0 if the hand is free, (ii) the values of the next p components are the names of the blocks immediately on table positions 1 through p, (iii) the values of the last n components are the names of the blocks immediately on top of blocks $a, b, c, \ldots$, where the value 0 means

"no block". For example, Figure 3 illustrates a state of the 4-Blocks World with 4 distinct table positions, encoded by

$$\langle c, 0, 0, b, d, 0, 0, 0, a \rangle .$$

in top presentation.

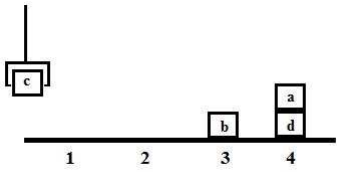

Figure 3: A state of the 4-Blocks World with 4 distinct table positions.

The state s^* has all blocks stacked up in increasing lexicographical order, starting with block a, on table position 1.

Consider the coarse abstraction $\Psi_{0,a}$, which keeps the name of the block a and the value 0 distinct, and maps all other constants to "don't care". The (non-spurious) abstract state corresponding to the reachable state depicted in Figure 3 would then be encoded as $\langle dc, 0, 0, dc, dc, 0, 0, 0, a \rangle$, where dc is for "don't care". The original state $s = \langle 0, a, 0, 0, 0, c, 0, d, b \rangle$ is reachable from the state in Figure 3: one places c first onto table position 2, then picks up a and places it on position 1, moves c to the top of a, and then stacks d and b onto c. Consequently, the abstract version $\Psi_{0,a}(s) = \langle 0, a, 0, 0, 0, dc, 0, dc, dc \rangle$ is not spurious. The operator

$$\langle 0, -, -, -, -, b, 0, -, - \rangle \rightarrow \langle b, -, -, -, -, 0, 0, -, - \rangle$$

picking up block b from the top of block a does not apply to s, but its abstract version, in which b is replaced by dc, does apply to $\Psi_{0,a}(s)$, producing the abstract state $\langle dc, a, 0, 0, 0, 0, 0, dc, dc \rangle$, which corresponds to Block a being the only block on the table, while there is no block on top of Block a. The latter state is clearly spurious—no reachable original state maps to it. $\Psi_{0,a}(s)$ will be generated by CA. Note, however, that although it is spurious, $\Psi_{0,a}(s)$ contains no mutex pairs involving the constants 0 and a, and therefore the generation of this state by CA will not result in CA mistakenly inferring that some mutex pair involving 0 and/or a is reachable.

In fact, when running CA on this domain with 9, 12, and 15 blocks, each with 3 table positions, we empirically verified that CA finds all mutex pairs.[3] This first of all provides proof of our claim above, stating that the condition in Theorem 1 is not necessary. Second, it suggests that conditions other than those given in Theorem 1 can yield a guarantee for CA's completeness.

A closer look at the top representation of the Blocks World with Table Positions reveals such a condition.[4] Every operator in this representation has two properties that help the CA method: (i) each operator swaps the values of two of the variables that are tested in its preconditions, all other values remain unchanged; and (ii) any variable not tested by the precondition of an operator can be assigned any value other than the ones swapped by the operator.

We can prove that these two properties are sufficient for CA to find all mutex pairs:

Observation 1 *Suppose each operator o in the domain representation satisfies the following two properties, where $(V_{i_1} = v_1, \ldots, V_{i_k} = v_k)$ are the preconditions of o.*

1. *The postconditions of o are of the form $(V_{i_1} = w_1, \ldots, V_{i_k} = w_k)$, where $w_i = v_i$ for all values of i except two, i' and i'', and for those indices $w_{i'} = v_{i''}$ and $w_{i''} = v_{i'}$. (o leaves the values of all variables other than $V_{i'}$ and $V_{i''}$ unchanged.)*
2. *Let W be a variable different from every V_{i_j} and w a value different from $v_{i'}$ and $v_{i''}$. Then $(W = w, V_{i_1} = v_1, \ldots, V_{i_k} = v_k)$ is contained in some reachable state.*

Then CA finds all mutex pairs.

Proof Let $v, w \in \Sigma$ and let t be any abstract state reachable from $\Psi_{v,w}(s^*)$ in the abstract state space created by the coarse abstraction $\Psi_{v,w}$. Suppose that t contains some pair of the form $(V = v, W = w)$. We will prove by induction on the abstract distance d of t from $\Psi_{v,w}(s^*)$ that $(V = v, W = w)$ is not mutex.

The base case, $d = 0$, occurs when $t = \Psi_{v,w}(s^*)$. Since s^* is reachable, any pair of the form $(V = v, W = w)$ in t is not mutex. Now suppose no abstract state at distance d from $\Psi_{v,w}(s^*)$ contains a mutex pair of the form $(V = v, W = w)$, and let t be at distance $d + 1$. Then there is an operator o whose abstract version $\Psi_{v,w}(o)$ maps an abstract state t' at distance d to t. The pair $(V = v, W = w)$ in t can be produced by $\Psi_{v,w}(o)$ in three different ways.

First, $(V = v, W = w)$ in t might be produced because $\Psi_{v,w}(o)$ does not change the values of V or W. In this case $(V = v, W = w)$ occurs in t' and so, by the inductive hypothesis, it is not mutex.

Second, $(V = v, W = w)$ in t might be produced because $\Psi_{v,w}(o)$ swaps the values of V and W. By property (1), then $(V = w, W = v)$ occurs in the preconditions of o and therefore, by property (2), there is at least one reachable state s in the original space to which o applies. In particular, in s we will have $(V = w, W = v)$, and therefore there will be a reachable state, $o(s)$, in which $(V = v, W = w)$. Therefore $(V = v, W = w)$ is not mutex.

[3] We verified this by manually calculating the actual number of mutex pairs using domain-specific knowledge and then comparing that number to the number of mutex pairs found by CA.

[4] In our representation, all operators have preconditions testing three variables: (a) what the hand is holding (must be 0 for pickup operators and must be the name of a block for put down operators); (b) what is in top of the block to be picked up or put down (must be 0); (c) what is on top of the block or table position from which the block is being picked up (must be the name of the block being picked up) or onto which it is being put down (must be 0). In other words, the preconditions of all operators test three variables: two of them must be 0 and the third must be the name of the block being moved. The operators swap what is in the hand with what is in the position onto which the block is being placed or from which it is being picked up.

The only other way to produce $(V = v, W = w)$ in t by $\Psi_{v,w}(o)$ is if one of these variables (say W) is unchanged by $\Psi_{v,w}(o)$ but the other one (V) is changed. This means V obtains its value of v in t by swapping its value, x, in t', with some variable X that has the value v in t'. We may assume x is not equal to v or w for otherwise one of the other two cases would apply with respect to the variables V and X. In other words, in t' we have $(X = v, V = x, W = w)$, with $x \notin \{v, w\}$, and the preconditions of o test $(X = v, V = x)$, and possibly other variables, but not necessarily $W = w$. By property (2), we know there is at least one reachable state s in the original space which satisfies $W = w$ in addition to all the preconditions of o. Therefore o will apply to s and $o(s)$ will have $(X = x, V = v, W = w)$. Therefore $(V = v, W = w)$ is not mutex.

We have thus shown that at no distance $d \geq 0$ is there a reachable abstract state containing a mutex pair, and therefore CA will not label any mutex pair as "non-mutex". It will therefore find all mutex pairs. ■

Note that even in cases when Observation 1 applies, a coarse abstraction may contain spurious states, but these spurious states will not contain any mutex pairs formed by the values kept distinct by the coarse abstraction. Observation 1 is presumably of less practical use than Theorem 1, because establishing the properties required by Observation 1 involves proving certain reachability conditions. We use this observation here simply to formally explain why CA is complete on the top representation of the Blocks World.

Theorem 5 *CA finds all mutex pairs in any Blocks World with Table Positions domain, represented in standard representation or in dual representation.*

Empirical Evaluation

We conducted two small sets of experiments. First, we tested CA on the domains described above, to verify that its running time was acceptable. Second, for some cases in which our formal guarantees on CA do not apply, we conducted a small experimental study to see whether CA can still be effective.

For the first experiment, we ran CA on the 5×5-Sliding-Tile Puzzle in both representations, on the 28-belt Scanalyzer, and on the 26-Blocks World with 3 Table Positions in top representation. With standard computers, CA takes on the order of a minute for the 5×5-Sliding-Tile Puzzle and about 2 minutes on the other two domains. This suggests that CA is very efficient. The reason is that in all these cases the number of states in every coarse abstraction is very small and therefore the amount of time required to enumerate a small number of coarse abstract state spaces is not large. For example, for the 28-belt Scanalyzer, there are 30 constant symbols (one for each belt and two for the analyzed status), resulting in $\binom{30}{2} + 30 = 465$ coarse abstractions. Most of these coarse abstractions have about 750 abstract states.

For the second experiment, we chose a different representation of the Blocks World with distinct table positions, as well as two new domains. The particular representations were intentionally chosen so that many mutex pairs exist; hence they are not necessarily the most natural or the most compact. It should be noted that Theorem 1 does not apply to any of the domains (in the chosen representations) in this experiment.

In our experiments, we fix the choice of the state s^* with respect to which we consider states and pairs reachable always to be the standard goal state.

Domain 1: Blocks World with Distinct Table Positions In an alternative representation of the n-Blocks World with p table positions, called the *height representation*, a state is encoded as a vector of length $1 + 3n + p$, where (i) the value of the first component is the name of the block in the hand or 0 if the hand is free, (ii) for every block a sequence of 3 components encodes the table position, the height of the block within a stack of blocks (where 1 means that the block is sitting directly on the table), and whether there is any block on top of this block (value 1) or not (value 0), (iii) the last p components are flags stating, for each table position, whether there are any blocks on top of it (value 1) or not (value 0). For example, the state in Figure 3 would be encoded as

$$\langle c, 4, 2, 0, 3, 1, 0, 0, 0, 0, 4, 1, 1, 0, 0, 1, 1 \rangle .$$

Domain 2: Towers of Hanoi In the n-Disks Towers of Hanoi with p Pegs, a state describes the constellation of n disks stacked on p named pegs. In every move, a disk can be transferred from one peg to another provided that all disks on the destination peg are larger than the moving disk.

In our representation of the n-Towers of Hanoi with p Pegs, called the *stack representation*, a state is encoded as a vector of length $p(n + 1)$, where for every peg a sequence of $n + 1$ components encodes the number of disks and the names of disks stacked on this peg (starting from the bottom of the peg); for a stack of k disks, the last $n - k$ components for this peg contain a 0 where the value 0 means "no disk". For example, Figure 4 illustrates a state of the 4-Disk Towers of Hanoi with 3 Pegs, encoded by $\langle 1, 4, 0, 0, 0;\ 1, 3, 0, 0, 0;\ 2, 2, 1, 0, 0 \rangle$ in the stack representation.

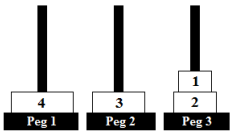

Figure 4: A state of the 4-Disk Towers of Hanoi with 3 Pegs.

This domain is a good example for our reason to choose seemingly "unnatural" domain representations. A "natural" approach for representing the n-Disk Towers of Hanoi with p Pegs would be to encode every state as a vector of n components. Each component corresponds to a disk; its value in $\{1, 2, ..., p\}$ represents the peg on which the disk is located. For example, the state in Figure 4 would be encoded as $\langle 3, 3, 2, 1 \rangle$. However, this domain representation does not result in any mutex pairs, because every one of the possible p^n vectors corresponds to a reachable original state. Hence this domain representation is not useful for our studies.

Domain 3: Constrained Sliding-Tile Puzzle We modified the Sliding-Tile Puzzle by disallowing some of the tile movements. Two versions of this "Constrained-Movement Sliding-Tile Puzzle" were used—see Figure 5—encoded in the same way as the standard representation of the original Sliding-Tile Puzzle. Note that, in both constrained versions, all the operators are invertible.

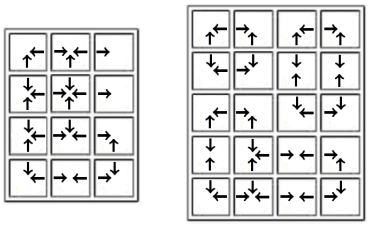

Figure 5: 3×4 and 4×5 Constrained-Movement Sliding-Tile Puzzle. The arrows indicate the possible movements of the tiles, based on the blank location.

Experimental Results

As shown above, CA is complete and very efficient for specific representations of many of the typical benchmark domains. However, in our further experiments, CA performed very poorly. CA does not find *any* mutex pairs on the 9-Blocks World with 3 Table Positions in the height representation (despite the fact that it is complete on the same domain in top representation) or on either version of the Constrained Sliding-Tile Puzzle. It finds some mutex pairs, but not all of them, for the Towers of Hanoi domain with 9 and 12 disks, each with 4 pegs.

The reason is *not* that mutex pairs are scarce in these domains: the 9-Blocks World with 3 Table Positions in height representation has 10,986 reachable pairs and 1,218 mutex pairs; the Towers of Hanoi with 9 disks on 4 pegs has 19,512 reachable pairs and 58,488 mutex pairs; the version with 12 disks has 50,627 reachable pairs and 177,467 mutex pairs, as we calculated using domain-specific knowledge.

We conclude that the success of CA depends on both the domain and its representation. It can fail badly, but it is the method of choice whenever the formal guarantee from Theorem 1 applies, since it is the only existing method that is provably guaranteed to be complete under known non-trivial circumstances.

Related Work

Mutex detection was originally suggested for finding pairwise conflicts between actions and between facts within Graphplan (Blum and Furst 1995). In Graphplan, two actions or two facts at the same level of reasoning are considered mutually exclusive if there is no valid plan containing both actions or making both facts true at the same time. Inspired by Graphplan's method, many planners have adopted some sort of mutex detection in their reasoning process, thereby achieving significant improvement in their performance. The methods range from being hand-coded for a domain to being domain-independent and fully automatic (Vidal and Geffner 2006), (Kautz and Selman 1998), (McCluskey and Porteous 1997). The systems SATPLAN04 (Kautz 2004), BLACKBOX (Kautz and Selman 1996), DISCOPLAN (Gerevini and Schubert 2000), MIPS (Edelkamp and Helmert 2001), MaxPlan (Chen, Xing, and Zhang 2007), FD (Helmert 2006), and those introduced in (Gerevini, Saetti, and Serina 2003) and (Penberthy and Weld 1994) are examples of planners using mutex detection.

State-of-the-art techniques for detecting mutexes in binary domain representation often use invariants. An invariant is a property that holds true in all reachable states of a state space and is usually detected by analyzing the problem description in a process called domain analysis. Examples of popular algorithms for deriving invariants are those used in the planners MIPS (Edelkamp and Helmert 2001) and FD (Helmert 2006), as well as those suggested by (Gerevini and Schubert 1998), (Scholz 2000), (Rintanen 2000), and (Fox and Long 1998). Rintanen's invariant synthesis algorithm generalizes the method used in HSP (Bonet and Geffner 1999). HSP is tailored to detect a certain type of mutex that Graphplan misses. Fox and Long (2000) propose a modification to Graphplan to detect this type of mutex as well. Constrained abstraction (Haslum, Bonet, and Geffner 2005) uses particular types of invariants to detect some mutexes in the description of an abstract state. For more background on domain analysis and examples of constrained abstractions using invariants the reader is referred to (Haslum 2006), where mutex pairs are discussed as a special case of "at-most-one" invariants consisting of only two atoms. In this work, Haslum also introduces the h^2 heuristic, which is a state-of-the-art method for finding mutex pairs.

In temporal planning (Gerevini, Saetti, and Serina 2003), one typically extends the notion of "mutex". While the classical mutex definition only refers to conflicts at the same time step, an extension to mutex detection is to look for conflicting actions and facts across different time steps, as implemented in MaxPlan (Chen, Xing, and Zhang 2007) and CPT (Vidal and Geffner 2006). Using domain-dependent knowledge, distance-based constraints were extracted and used in CPT. Here, if a pair of constraints is mutex, a distance value indicates the minimum number of reasoning steps that must lie in between them. LONDEX (used in MaxPlan) was later proposed for automatically detecting such distance-based constraints. Another feature of LONDEX constraints is that they are extracted from a multi-valued domain formulation (MDF) (Helmert 2006) of a problem domain, using domain transition graphs.

We compared the performance of LONDEX on a binary representation of the Sliding-Tile Puzzle to that of CA on the two multi-valued representations presented above. For any version of the puzzle of size $n \times \ell$, for $n, \ell \geq 2$ and $n \times \ell \geq 5$, LONDEX always finds only half of the mutex pairs (e.g., 22,680 of 45,360 for the 6×6-puzzle). Our CA approach is known to be complete in this case in both multi-valued representations, i.e., it finds all mutexes.

Mutex detection methods usually achieve efficiency at the cost of missing mutex constraints. For example, since Bonet and Geffner's (1999) algorithm works on grounded representations, in order to make it practical, the search for mutex

pairs is systematically limited to a restricted class. Graphplan produces a different, yet still incomplete set of mutex constraints. The algorithm by Gerevini and Schubert (1998) generates many classes of invariants in addition to mutexes, but at the cost of decreasing the performance.

Conclusions

We introduced CA, a mutex detection method applicable to multi-valued domain representations, which is guaranteed to detect all mutex pairs under certain natural and easy-to-check conditions. CA is based on exhaustive search in tiny abstractions of the original state space, which makes it a simple and efficient approach.

While we formulated and evaluated CA only for mutex pairs, its fundamental idea can be used to detect mutexes of higher order as well (e.g., triples of variable-value assignments that do not occur in any reachable state)—our formal guarantee of completeness given by Theorem 1 will still apply. It remains to be tested to which order of mutexes the approach scales. Another issue of future research is to find further natural and easy to check conditions under which CA is provably complete.

References

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1):1636–1642.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of ECP-99*, 360–372. Springer.

Chen, Y.; Xing, Z.; and Zhang, W. 2007. Long-distance mutual exclusion for propositional planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1840–1845.

Edelkamp, S., and Helmert, M. 2001. MIPS: The Model-Checking Integrated Planning System. *AI Magazine* 22(3):67–72.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *J. Artif. Intell. Res.* 9:367–421.

Gerevini, A., and Schubert, L. K. 1998. Inferring state constraints for domain-independent planning. In *AAAI/IAAI*, 905–912.

Gerevini, A., and Schubert, L. K. 2000. Discovering state constraints in DISCOPLAN: Some new results. In *AAAI/IAAI*, 761–767.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *J. Artif. Int. Res.* 20:239–290.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence*, 1163–1168.

Haslum, P. 2006. *Admissible Heuristics for Automated Planning*. Linköping Studies in Science and Technology: Dissertations. Department of Computer and Information Science, Linköpings Universitet.

Helmert, M., and Lasinger, H. 2010. The Scanalyzer domain: Greenhouse logistics as a planning problem. In *ICAPS*, 234–237.

Helmert, M. 2006. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

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

Johnson, W. W., and Story, W. E. 1879. Notes on the "15" puzzle. *American Journal of Mathematics* 2(4):397–404.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. 1194–1201. AAAI Press.

Kautz, H. A., and Selman, B. 1998. The role of domain-specific knowledge in the planning as satisfiability framework. In *AIPS*, 181–189.

Kautz, H. 2004. SATPLAN04: Planning as satisfiability. In *In Proceedings of IPC4, ICAPS*.

McCluskey, T. L., and Porteous, J. M. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artif. Intell.* 95(1):1–65.

Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In *AAAI*, 1010–1015.

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on on Innovative Applications of Artificial Intelligence*, 806–811.

Scholz, U. 2000. Extracting state constraints from pddl-like planning domains. In *Proceedings of the AIPS Workshop on Analyzing and Exploiting Domain Knowledge for Efficient Planning*, 43–48.

Solèr, M. 2012. Refining abstraction heuristics with mutexes, Bachelor thesis, University of Basel.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artif. Intell.* 170:298–335.

Zilles, S., and Holte, R. C. 2010. The computational complexity of avoiding spurious states in state space abstraction. *Artificial Intelligence* 174:1072–1092.