Detecting Mutex Pairs in State Spaces by Sampling

Mehdi Sadeqi[1], Robert C. Holte[2], and Sandra Zilles[1]

[1] Department of Computer Science, University of Regina
Regina, SK, Canada S4S 0A2, {sadeqi2m|zilles}@cs.uregina.ca
[2] Department of Computing Science, University of Alberta
Edmonton, AB, Canada T6G 2E8, holte@cs.ualberta.ca

Abstract. In the context of state space planning, a mutex pair is a pair of variable-value assignments 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 Missing Mass Method (MMM)—a new efficient and domain-independent method for mutex pair detection, based on sampling reachable states. We exploit a recent result from statistical theory, proven by Berend and Kontorovich in [1], that bounds the probability mass of missing events in a sample of a given size. We tested MMM empirically on various sizes of four standard benchmark domains from the planning and heuristic search literature. In many cases, MMM works perfectly, i.e., finds all and only the mutex pairs. In the other cases, it is near-perfect: it correctly labels all mutex pairs and more than 99.99% of all non-mutex pairs.

1 Introduction

The aim of heuristic search and planning systems is to find a path (sometimes a least-cost path) from a given start state to a given goal using a given set of operators. The set of possible states is defined by specifying a set of variables and the possible values for each variable, and a particular state is specified by assigning a specific value to each variable. For example, in the familiar 8-puzzle (3×3-sliding-tile puzzle), there might be 9 variables, one for each position of the puzzle (e.g. variable UL might refer to the upper-left corner position), and the value of each variable indicates which tile (or "no tile") is in that position. We use the phrase "variable-value assignment" to refer to the assignment of a specific value to a specific variable, for example, UL="no tile" is a variable-value assignment. Some planning systems use propositional variables which we treat as variables that can take on one of two values (true and false).

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 valid state. By "valid state" one typically means a state that is reachable from a given start state. For instance, in the 8-puzzle, an example of a mutex pair would be (UL="no tile" and BR="no tile"), where BR is the variable saying what is in

the bottom-right position. In the 8-puzzle, there is only one empty position, so there cannot simultaneously be two variables that have the value "no tile".

Mutex pair detection was first addressed by Dawson and Siklóssy [6], and has been in use for improving the performance of planning systems since the development of Graphplan [2]. In Graphplan, two actions or two facts at the same level of reasoning are mutex if there is no valid plan allowing both actions or both facts at the same time. The original success of mutex detection in Graphplan [2] led to the development of many efficient planners using mutex detection, such as temporal planners, different SAT-based planners and planners based on constraint programming. State-of-the-art techniques for detecting mutexes in binary domain representation often use invariants, i.e., properties that hold true in all reachable states of a state space. Using an invariant synthesis algorithm, HSP [3] is tailored to detect a certain type of mutex that Graphplan misses. Constrained abstraction [8] 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 [7], 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.

One use of mutex detection is to translate propositionally encoded planning domains into representations with multi-valued state variables; e.g., FD [9] can thus reveal intuitive dependencies between variables which helps to explicate some of the implicit constraints of propositional planning tasks.

Another application of mutex detection is search space pruning. In regression planning, nodes in the search space are partial assignments of state variables, and edges are actions. Search proceeds backward from the goal until reaching a node consistent with the start state. To reduce search effort, one prunes nodes that contain the assignment of some previously encountered node. Further, one often prunes nodes that are "impossible" because they contain mutex pairs.

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, most notably pattern databases (PDBs) [5], 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 [15]. 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 heuristics, and thus to speed up search.

Unfortunately, there are no known efficient methods for detecting *all* mutex pairs. Existing algorithms usually make a compromise in the number of detected mutex constraints for the computational complexity of the algorithm. Various methods differ in the number and type of mutex constraints they detect.

In this paper, we propose the Missing Mass Method (MMM)—a new algorithm for detecting mutex pairs, based on sampling reachable states. We exploit a recent result from statistical theory, proven by Berend and Kontorovich in [1], that would allow us to bound the probability of missing a reachable pair (i.e., a non-mutex pair) in an i.i.d. sample of a given size. The main advantages of MMM over existing methods are the following.

- It is very simple to describe and to implement.
- As opposed to many state-of-the-art mutex detection techniques, MMM does not systematically restrict itself to detecting only a subclass of the mutex pairs.
- For several standard benchmark domains, MMM is *perfect* on reasonable domain sizes, i.e., it detects mutex pairs with 100% accuracy. For the same domains, it scales very well, mostly yielding perfect accuracy even for very large domain sizes, e.g., for Scanalyzer with 100 batches, Blocks World with 26 blocks, or the 10×10-Sliding-Tile Puzzle.
- While most of our experiments on MMM were on detecting mutex pairs, exactly the same method can be used to detect higher order mutexes. For instance, a mutex of order 3 in the Blocks World would be a triple of variable-value assignments that represents the facts *Block a is on top of Block b, Block b is on top of Block c*, and *Block c is on top of Block a*.
- MMM does not require backward reasoning, depending on the application for which it is used. Technically, MMM can be used with any kind of sampling method. (Theoretical guarantees only hold though for special sampling methods that will typically not be available in practice.)

All existing mutex detection methods err on one side: they might consider mutex pairs as reachable but they will never flag a reachable pair as mutex. As opposed to that, our method errs on the other side. It never considers a mutex pair as reachable, but it may consider reachable pairs as mutex. Depending on the application, this might be an advantage or disadvantage— we will discuss that below. Our experimental results will illustrate that MMM is very reliable in a large variety of state spaces, suggesting that it can be safely applied even in cases when it is crucial not to consider reachable pairs as mutex. In some such cases, we will demonstrate empirically that MMM can clearly outperform state-of-the-art mutex detection methods.

2 The Missing Mass Method for Mutex Detection

Assume that states are represented as variable-value pairs in m variables. We denote the variables with $x_1, \ldots, x_m$, so that the state vector $(a_1, \ldots, a_m)$ cor-

responds to the assignment vector $(x_1 = a_1, \ldots, x_m = a_m)$. 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.*

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 a_i, a_j, the partial original state $(x_i = a_i, x_j = a_j)$ is a reachable pair if there are a_k, for $k \in \{1, \ldots, m\} \setminus \{i, j\}$ such that $(a_1, \ldots, a_m)$ is a reachable state; otherwise $(x_i = a_i, x_j = a_j)$ is a mutex pair.

Our approach to detecting mutex pairs in large problem domains is based on sampling. The general scheme of our method, which we will call the *Missing Mass Method (MMM)* for reasons detailed below, is quite simple:

1. Determine a number N of pairs of variable-value assignments to be sampled.
2. Determine the smallest integer N_s such that $N_s \cdot \binom{m}{2} \geq N$. (Thus, sampling N_s states results in sampling at least N pairs of variable-value assignments.)
3. Sample N_s reachable states and extract all pairs of variable-value assignments from them.
4. Any pair of variable-value assignments not encountered this way is considered a mutex pair.

The two details that need to be defined are (i) how the sampling of reachable states is done, and (ii) how to fix the number N of pairs of variable-value assignments to be sampled.

Concerning (i), let us first assume we have fixed a method for sampling reachable states, which induces a probability distribution D over all possible pairs of variable-value assignments. (We experiment with a variety of sampling methods, as described in the following section.)

Statistical theory provides us with tools for addressing question (ii) under these circumstances. Berend and Kontorovich [1] give an upper bound on the expected probability mass of the elements not seen after taking N i.i.d. samples from any fixed distribution. This bound depends on the total number z of elements in the (finite) universe and is given by the following inequality.

$$\mathbb{E}_D[M_N] \leq \frac{z}{eN} \text{ for } N > z\,. \tag{1}$$

Here M_N is the total probability mass of the elements not seen after sampling N times from the distribution D that results from the sampling method (called the *sampling distribution*), and $\mathbb{E}_D[M_N]$ is its expected value with respect to D.

We deploy this bound by choosing a number N of samples that is large enough for $\mathbb{E}_D[M_N]$ to be below a fixed threshold. The required number z may not be available, but an upper bound on z is obtained by computing the total number of possible pairs of variable-value assignments in the given representation of the state space. For example, if each of the m state variables can take one of

k possible values, then there are $\binom{m}{2}$ many variable pairs, each with k^2 many possible value assignments, and thus a total of $\binom{m}{2}k^2$ possible pairs of variable-value assignments. We use this upper bound of $\binom{m}{2}k^2$ in lieu of z, since this never makes us sample less than when using the exact value of z.

Given this method for computing the sample size, we still need to fix a method for choosing the samples. The sampling process itself may be of crucial importance to the success of the scheme depicted above. Inequality (1) does not provide us with a sample size that bounds the probability of missing a pair, but with a sample size that bounds the probability mass of the non-seen pairs with respect to the sampling distribution. If some pairs of variable-value assignments have too small a probability of being sampled, then even with a sample resulting from a very low threshold for $\mathbb{E}_D[M_N]$ we might be missing these reachable pairs, because their cumulative probability mass with respect to the sampling distribution is too small. Consequently, if a poor sampling method is used, MMM might flag reachable pairs as mutex.

Designating reachable pairs as mutex could have positive or negative effects, depending on the problem that mutex detection is applied to. Hence, to minimize the risk of missing reachable pairs, it is desirable to use a near-uniform sampling process, so that no pairs of variable-value assignments have too small a probability of being sampled. Unfortunately, there is no known method for sampling states in a way that creates a near-uniform sample of the contained pairs of variable-value assignments, and further one does in general not sample pairs i.i.d. when sampling states. Because of the latter problem, Berend and Kontorovich's bound does not even yield theoretical guarantees in our case. We nevertheless use their bound to decide how many states to sample. Note that even with uniform sampling there would be no *guarantee* that our method finds all reachable pairs; however, we would have a guaranteed minimum probability of finding all reachable pairs. Our experiments suggest that for typical benchmark domains, even in large sizes, this probability is very high. In our experiments we tested a variety of sampling methods, which we will describe in Section 3.1.

2.1 Does MMM err on the wrong side?

An important property distinguishing MMM from existing mutex detection methods is that it errs "on the other side". While existing methods never consider a reachable pair mutex, MMM never considers a mutex pair reachable, but might consider a reachable pair mutex.

In the case of state space pruning in regression planning, falsely considering a reachable pair as mutex might lead to the elimination of reachable states and thus to the elimination of paths from goal to start, in the worst case disconnecting the goal from the start state. The effect could be devastating, but a closer look at our experimental results will show that in many cases this would not be a major concern when applying MMM.

Considering the problem of improving heuristic values when a PDB contains spurious abstract states, the one-sided error of MMM may even have a positive

effect. To address this problem with MMM, one (i) builds a PDB as usual, then (ii) uses MMM backwards from the goal in the original state space to find mutex pairs, (iii) builds an auxiliary PDB in the usual way with the exception that abstract states containing a mutex pair are not added to the open list, but are considered deadends. (This means one will find paths to abstract states that do not pass through abstract states suspected of being spurious.) Finally, (iv) one replaces entries in the PDB with those from the auxiliary PDB as long as the latter are not infinite. The resulting PDB would be used to guide the search. Since our method can flag reachable pairs as mutex, the resulting heuristic might be inadmissible and inconsistent, causing A* or IDA* to find only suboptimal solutions, but it may potentially find them much faster than with an admissible heuristic obtained before removing abstract states that contain mutexes.[3] The reader is referred to [14] for recent work on efficient suboptimal search with inadmissible heuristics.

We claim that MMM will be a very useful method for mutex detection, mainly because our experimental results demonstrate that for a large variety of state spaces, MMM is 100% accurate (it does not flag any reachable states mutex). In some cases, it outperforms all existing methods in terms of accuracy, while still being very efficient. The trade-off between efficiency and accuracy seems to be much less substantial for MMM than it is for existing mutex detection methods. (For a comparison of MMM with h^2 in terms of accuracy, see Section 4.)

3 Experimental Setup

Four planning and search benchmark problem domains, represented using production system vector notation (PSVN) [11], were selected for this study. All operators in all domain representations are invertible. While we describe the representations of the domains below, we omit a general description of the domains themselves, due to space constraints.

The particular representations were intentionally chosen so that many mutex pairs exist; hence they are not necessarily the most natural or the most compact.

In our experiments, we fix the choice of the state s^* with respect to which we consider states reachable, instead of running MMM on a variety of choices for s^*. In all the spaces we use for testing, operators are invertible, so that within any connected component, every state is reachable from every other state. Hence, the set of reachable pairs is the same no matter which s^* in a connected component we use. However, the distance from one choice of s^* to all the reachable pairs might be different than the distance from another s^*, and that could affect the success rate of some of the sampling methods. We believe that the standard goal state for each space is a good, representative choice for s^*, to measure the

[3] We have initial empirical results supporting this claim for one representation of the Towers-of-Hanoi domain. Further, the described method can be implemented by flagging states considered spurious in the original PDB without actually building a separate PDB.

success of a sampling method. Hence, for any domain, we always chose s^* to be the standard goal state.

Domain 1: Towers of Hanoi. We encode a state of the n-Disks Towers of Hanoi with p pegs 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.

This domain illustrates why we choose seemingly "unnatural" domain representations. A "natural" approach for representating 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. However, this representation does not yield 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. (In addition, the number of operators in the "natural" representation described above is also exponential in the number of disks when $p > 3$, making this representation inconvenient for other reasons as well.)

Domain 2: Blocks World with Table Positions. We consider two PSVN representations of the n-Blocks World with p named table positions. In the first one, called the *top representation*, a state vector has $1 + p + n$ components, each containing either the values 0 or one of n possible block names: (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 $a, b, c, \ldots$.

In the *stack representation*, a state is encoded as a vector of length $p(n+1)+1$, where for every table position a sequence of $n+1$ components encodes the number of blocks and the names of blocks stacked on this position (starting from the bottom); for a stack of k blocks, the last $n-k$ components for this block contain a 0. The final component encodes the content of the hand. Note the similarity of this domain in this representation to our representation of the Towers of Hanoi.

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

Domain 3: Sliding-Tile Puzzle. In the *standard representation* of $n \times \ell$-Sliding-Tile Puzzle, states are represented as vectors of length $n \cdot \ell$, where each component corresponds to 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). In the *dual representation*, 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.

The state s^* 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.

Domain 4: Scanalyzer. In the PSVN representation of the n-Belt Scanalyzer [10] (for even n), 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. The state s^* corresponds to having all plant batches analyzed and placed on their original conveyor belts.

3.1 Sampling

In our experiments, we used a fixed threshold of 0.00001 by which to bound the estimated missing probability mass. For example, for the 4×5-puzzle in standard representation, the resulting number of pairs to be sampled was $2,795,883,753$. Since there are 190 pairs of variable-value assignments in every state, this corresponds to sampling $14,715,178$ states (note again that the pairs are then not sampled i.i.d., but our experiments will show that the missing mass bound is still effective). Similar calculations for various sizes of the domains we experimented with shows this approach to be scalable (e.g., the number of states to be sampled for the 10×10-puzzle is $367,879,441$). In other words, the sample size used in our approach is small enough for our sampling method to be feasible.

We tested a variety of sampling processes, in particular we report our experiments on single random walks (RW), and Frontier Sampling (FS) [12], always beginning at s^*. Furthermore, we tested uniform sampling of reachable states (USS). Note that USS does not necessarily mean uniform sampling of the reachable pairs of variable-value assignments. While USS is not a domain-independent method, RW and FS are.

For RW we conducted basic random walks without parent pruning and without restarts; thus the length of a random walk was the number of states we wanted to sample. FS conducts r dependent random walks in a search tree by keeping a list of r nodes [12]. Initially, some r nodes are sampled at random. From the joint list of all children of these r nodes, one child c is chosen uniformly at random as the next sampled node; c then replaces its parent in the list of r nodes. In our experiments, we set $r = 100$; the initial r nodes are chosen by 100 independent random walks conducted from s^*. The lengths of these random walks are chosen uniformly at random from $\{0, 1, \ldots, 1000\}$.

4 Experimental Results

We first tried MMM on small sizes of the domains described above. Here we enumerated the state space exhaustively and directly compared the true set of reachable pairs to those found by MMM, which, using any of USS, FS, and RW, found all reachable pairs and thus was perfect at mutex pair detection. For larger domains, we calculated the actual number of reachable pairs for every representation and compared it to the number of pairs MMM found.

For illustration, we show how to compute the actual number of reachable pairs for the 28-belt Scanalyzer, which is $331,184$. In the representation we use, any batch can occur in any location independent of where any other batch is

located; any batch can be analyzed/not-analyzed independent of the status of any other batch; and the location of any batch is independent of the anlayzed status of any batch (including itself). Thus a pair of variables can take one of three possible forms: (i) Both variables represent batches; there are $28 \cdot 27/2$ such pairs of variables, each with $28 \cdot 27$ possible value assignments, resulting in $285,768 = (28 \cdot 27/2) \cdot 28 \cdot 27$ reachable pairs. (ii) Both variables represent an "analyzed" status; there are $28 \cdot 27/2$ such pairs of variables, each with $2 \cdot 2$ possible value assignments, resulting in $1,512$ reachable pairs. (iii) One variable represents a batch, the other represents an "analyzed" status. This results in $43,904 = (28 \cdot 28) \cdot 28 \cdot 2$ pairs. These numbers sum up to $331,184$. We do not show the calculations for the other domains, but report the resulting numbers below. Note that we do not use these numbers to calculate the bound in Inequality (1); instead in our experiments we assume that we only have the knowledge of how to compute the total number of pairs of variable-value assignments that can be expressed in the given domain representation language.

With any of the sampling methods USS, FS, and RW, MMM was perfect in the following testbeds: Blocks World with 12, 15, 18, 21, and 26 blocks, for 3 table positions in the top representation; 5×5-, 5×6-, and 10×10-Sliding-Tile Puzzle in the standard and 5×5-Sliding-Tile Puzzle in the dual representation; Scanalyzer with 12, 16, and 20 belts. For Scanalyzer with 28 and 100 belts, FS and USS were perfect, while RW missed a very small percentage of the reachable pairs. For Towers of Hanoi with 12 disks on 4 pegs, USS and RW were perfect, while FS missed a very small percentage of pairs. All methods missed a few pairs for the 26-Blocks World with 3 table positions in the stack representation.

Table 1 gives a representative sample of our results on larger domain versions. For each domain (in a particular representation and size), it shows the actual number of reachable pairs ("#Pairs"), the number of samples suggested by Inequality (1) ("Bound"), the percentage of reachable pairs that remained undetected after sampling as many states as suggested by the bound ("Missing Pairs"), and the actual number of samples after which all reachable pairs were found ("Minimum Sample Size"). The "Missing Pairs" number is averaged over 1000 repetitions of the whole sampling process. The "Minimum Sample Size" was obtained by repeating the whole sampling process 1000 times and recording the smallest multiple α of 100,000 such that all reachable pairs were always found when sampling α many states. The domains are the 12-disk Towers of Hanoi (ToH) with 4 pegs, the 26-Blocks World (BW) with 3 table positions, both in top and in stack (stk) representation, the 5×5-Sliding-Tile Puzzle (SP), both in standard (std) and in dual (du) representation, the 10×10-Sliding-Tile Puzzle in standard representation, and Scanalyzer (SCN) with 28 and 100 belts. Missing entries for RW on the Scanalyzer domain mean that even after sampling 100,000,000 (for 28 belts) and 1,000,000,000 (for 100 belts) reachable states, not all reachable pairs had been found.

MMM is very efficient in terms of running time. On a standard modern computer, MMM with FS for 1,000,000 sampled states takes on the order of one second for the 5×5-Sliding-Tile Puzzle in standard representation, less than one

Domain	#Pairs	Bound	Missing Pairs			Minimum Sample Size		
			USS	FS	RW	USS	FS	RW
ToH (12,4)	51,642	6,217,162	0	0.005%	0	4,000	10,000	6,000
BW top (26,3)	285,551	26,818,411	0	0	0	30	2,200	2,700
BW stk (26,3)	1,547,049	26,818,411	0.0001%	0.006%	0.005%	35,000	70,000	70,000
SP std (5×5)	180,000	22,992,465	0	0	0	14	800	800
SP du (5×5)	180,000	22,992,465	0	0	0	14	900	1,100
SP std (10×10)	49,005,000	367,879,441	0	0	0	300	12,000	12,000
SCN (28)	331,184	33,261,751	0	0	0.004%	16	200	-
SCN (100)	51,024,800	382,890,408	0	0	0.0007%	300	7,000	-

Table 1. MMM results for three different sampling methods, using a threshold of $0.00001 \geq \mathbb{E}_D[M_N]$. Minimum Sample Sizes are given as multiples of 1,000.

minute for the 10×10-Sliding-Tile Puzzle in standard representation or for the 28-belt Scanalyzer, and a few minutes for the 100-belt Scanalyzer. When taking as many samples as suggested by the bound, this would result in a time of less than half a minute for the 5×5-Sliding-Tile Puzzle (std), about half an hour for the 28-belt Scanalyzer, and a few hours for the 10×10-Sliding-Tile Puzzle (std). For the 100-belt Scanalyzer, sampling the full number suggested by the bound would take on the order of 1 day, but after less than half an hour actually all reachable pairs would have been found.[4] In general, the difference between the bound and the minimum sample sizes suggest that, in practical applications, one may set the threshold for $\mathbb{E}_D[M_N]$ substantially higher than 0.00001 and still obtain perfect results in many domains.

USS seems to be perfect whenever a feasible such sampling method exists, with the exception of Blocks World in the stack representation, where it misses on average 2 out of 1,547,049 reachable pairs. In many cases when a uniform method of sampling reachable states is not available, we can still expect that FS will work, though it probably needs to sample more before finding all reachable pairs and thus might not scale as well as USS. The simplest sampling method we tried, RW, works remarkably well. It is perfect on the top representation of Blocks World and on the Sliding-Tile Puzzle, and misses on average only 82 out of the 1,547,049 reachable pairs in the stack representation of Blocks World, 13 out of the 331,184 reachable pairs for the 28-belt Scanalyzer, and 350 out of the 51,024,800 reachable pairs for the 100-belt Scanalyzer.

When RW was missing a few pairs in the Scanalyzer domains, increasing the number of samples well beyond the bound (100,000,000 for 28 belts and 1,000,000,000 for 100 belts) still did not make the sampling perfect. The latter indicates that in these cases there were reachable pairs whose probability of being sampled by RW is so low that their cumulative weight under the probability distribution induced by the sampling procedure lies well below our threshold of 0.00001. Note that in some cases the percentages of missing pairs are slightly

[4] The time used for mutex detection could be amortized when solving a large number of search or planning problem instances in the same reachable component of the state space. Further, to the best of our knowledge, there are no methods that can solve an average instance of the 100-belt Scanalyzer in time on the order of a day.

higher than $0.00001 = 0.001\%$, but the cumulative probability of the pairs under the distribution resulting from the sampling may still be lower than 0.00001.

4.1 Comparison with h^2

h^2 is a state-of-the-art method for mutex pair detection in planning [7] that errs on the opposite side when compared to MMM, i.e., it will never consider a reachable pair mutex. h^2 is very effective for a large number of domains, for example, it perfectly detects all mutex pairs for Scanalyzer, the Blocks World with Table Positions, and for almost all sizes of the Sliding-Tile Puzzle, in the representations we experimented with.

However, it systematically fails to detect mutexes of certain types in some domains in which MMM is perfect or almost perfect. Firstly, in the 2×2-Sliding-Tile Puzzle, h^2 misses a special kind of mutex pair, namely some of the pairs that state that two specific distinct tiles reside in two specific distinct locations. This amounts to 20% (12 out of 60) of the existing mutex pairs being missed by h^2. Such pairs are never mutex in larger versions of the puzzle [13]. Secondly, in the stack representation of the n-Blocks World with p table positions, h^2 will systematically miss all mutex pairs that state, for some $i < n$ and some $j > i$, that $n-i$ blocks are on position a while a specific block is at height j on position $b \neq a$. Such a pair of variable-value assignments is mutex as it would require the existence of more than n blocks. For $n = 26$ and $p = 3$, h^2 misses 81% (711,291 out of 873,960) of the existing mutex pairs. Similarly, in our representation of the n-Disk Towers of Hanoi with p pegs, h^2 will miss all mutex pairs stating that there are $n - i$ disks on peg a while a specific disk is at height j on peg $b \neq a$.

This demonstrates that in many domains MMM has an advantage due to sampling at random as opposed to being constructed in a way that systematically misses certain types of mutex pairs.

5 Conclusions

We presented MMM, a sampling-based approach for detecting mutually exclusive pairs of variable-value assignments that is applicable to any kind of domain representation. The method is easy to implement, very efficient, and, if a reasonably good sampling procedure is used, also very effective in detecting mutexes. It does not systematically restrict itself to detecting only a subclass of the mutex pairs, finds mutexes with perfect accuracy in almost all of the domains we tested, and is very near perfect in all other domains tested, when using either of the two domain-independent sampling method we tried (FS, RW). MMM scales very well to large domain sizes. Initial empirical results (not reported here) suggest that MMM may even be successful at detecting higher-order mutexes.

We have demonstrated that h^2, a state-of-the-art mutex detection method, systematically misses certain mutex pairs in some domains. In a small experiment on binary domain representations (not reported here), we showed that the same is true for LONDEX [4] , another state-of-the-art method. MMM outperforms both

methods in the domains we experimented with, thanks to not being restricted to specific classes of mutex pairs a priori. In fact, all mutex detection methods we know of suffer from systematically missing certain pairs, and we are aware of only one method for which in some special kinds of domain completeness has been proven to be guaranteed (the CA method, see [13]).

Overall, we believe that MMM can improve the performance of planning systems and heuristic search without affecting runtime efficiency. MMM might further be of use when combined with a method that errs on the opposite side.

Acknowledgements. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Berend, D., Kontorovich, A.: The missing mass problem. Stat. and Prob. Lett. **82** (2012) 1102–1110
2. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1) (1995) 1636–1642
3. Bonet, B., Geffner, H.: Planning as heuristic search. Artif. Intell. **129**(1-2) (2001) 5–33
4. Chen, Y., Xing, Z., Zhang, W.: Long-distance mutual exclusion for propositional planning. In: IJCAI. (2007) 1840–1845
5. Culberson, J., Schaeffer, J.: Pattern databases. Comput. Intell. **14**(3) (1998) 318–334
6. Dawson, C., Siklóssy, L.: The role of preprocessing in problem solving systems. In: IJCAI. (1977) 465–471
7. Haslum, P.: Admissible Heuristics for Automated Planning. Linköping Studies in Science and Technology: Dissertations. Dept. of Computer and Information Science, Linköpings Univ. (2006)
8. Haslum, P., Bonet, B., Geffner, H.: New admissible heuristics for domain-independent planning. In: AAAI. (2005) 1163–1168
9. Helmert, M.: The Fast Downward planning system. J. Artif. Intell. Res. **26** (2006) 191–246
10. Helmert, M., Lasinger, H.: The Scanalyzer domain: Greenhouse logistics as a planning problem. In: ICAPS. (2010) 234–237
11. Hernádvölgyi, I., Holte, R.: PSVN: A vector representation for production systems. Technical Report TR-99-04, Dept. of Computer Science, Univ. of Ottawa (1999)
12. Ribeiro, B.F., Towsley, D.F.: Estimating and sampling graphs with multidimensional random walks. CoRR **abs/1002.1751** (2010)
13. Sadeqi, M., Holte, R.C., Zilles, S.: Using coarse state space abstractions to detect mutex pairs. In: SARA. (2013) 104–111
14. Thayer, J., Ruml, W.: Bounded suboptimal search: A direct approach using inadmissible estimates. In: IJCAI 2011. (2011) 674–679
15. Zilles, S., Holte, R.C.: The computational complexity of avoiding spurious states in state space abstraction. Artif. Intell. **174** (2010) 1072–1092