A Comparison of h^2 and MMM for Mutex Pair Detection Applied to Pattern Databases

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 state space search or planning, a pair of variable-value assignments that does not occur in any reachable state is considered a mutually exclusive (mutex) pair. To improve the efficiency of planners, the problem of detecting such pairs has been addressed frequently in the planning literature. No known efficient method for detecting mutex pairs is able to find all such pairs. Hence, the number and type of mutex constraints detected by various algorithms are different from one another. The purpose of this paper is to study the effects on search performance when errors are made by the mutex detection method that is informing the construction of a pattern database (PDB). PDBs are deployed for creating heuristic functions that are then used to guide search. We consider two mutex detection methods, h^2, which can fail to recognize a mutex pair but never regards a reachable pair as mutex, and the sampling-based method MMM, which makes the opposite type of error. Both methods are very often perfect, i.e. they exactly identify which pairs are mutex and which are reachable. In the cases that they err that we examine in this paper, h^2's errors cause search to be moderately slower (7%−24%) whereas MMM's errors have very little effect on search speed or suboptimality, even when its sample size is quite small.

1 Introduction

In heuristic state space search and planning, the goal is to find a (preferably optimal, i.e., least-cost) path from a start state to a goal. The state space is usually determined implicitly using a set of operators defining the problem domain. The states in this state space are specified using a set of variables with every variable having a set of possible values. A pair of variable-value assignments that do not co-occur in any reachable state is considered to be a "mutually exclusive pair" or mutex pair for short. For example, consider a representation of the well-known 8-puzzle (3×3-sliding-tile puzzle) where each position of the puzzle corresponds to one variable representing the tile in that location (e.g. variable UL might correspond to the upper-left corner position representing the numbered tile or "no tile" in that position). An example of mutex pair here would be (UL="no tile" and BR="no tile"), where BR is the variable refers to the bottom-right position

indicating what is in that position. Since there is only one empty position in this puzzle, these two variables cannot simultaneously have the value "no tile".

Mutex pair detection can be used for improving the performance of planning systems as was first shown in Graphplan [2]. In this planner, if there is no valid plan that allows two actions or two facts at the same level of reasoning, the latter are considered mutex. This motivated the adoption or development of mutex detection methods in other planners [3, 4, 6–10, 13, 14, 17–20, 23, 24].

h^2 is a state-of-the-art method for mutex pair detection in planning [12]. It is a conservative approach to mutex detection, i.e., it might consider mutex pairs as reachable but 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 almost all sizes of the Sliding-Tile Puzzle [22]. Despite its effectiveness, this method may fail to detect all mutex pairs. This happens, for example, for a special kind of mutex pair in the 2×2-Sliding-Tile Puzzle and in the so-called stack representation of the Blocks World with n blocks and p table positions. In the 2×2-Sliding-Tile Puzzle, some of the mutex pairs that state that two specific distinct tiles reside in two specific distinct locations are missed by h^2. This only happens in this very small version of the puzzle since such pairs are never mutex in larger versions of the puzzle [22]. In the stack representation of the n-Blocks World with p table positions, h^2 fails to detect all mutex pairs that state, for some $i < n$ and some $j > i$, that $n - i$ blocks are stacked on table position a while a specific block is at height j on table position $b \neq a$. This kind of variable-value assignment pairs is mutex since it entails the existence of more than n blocks [21].

A recently introduced mutex detection method is MMM [21]. Unlike all other approaches, MMM is not conservative, it errs "on the other side." It never considers a mutex pair reachable, but might consider a reachable pair mutex.

Mutex detection has several applications, most notably search space pruning. As an example, consider the search space of regression planning where nodes are partial assignments of state variables and edges are actions. Backward search starts from the goal and stops when we reach an assignment of state variables consistent with the start state. One way to reduce search effort is to remove nodes that contain mutex pairs. Another application of mutex detection is explicating some of the implicit constraints of propositional planning tasks. This is done by translating propositional planning domains into representations with multi-valued state variables and revealing dependencies between variables.

Mutex detection can also be used for improving abstraction-derived heuristic functions. These are functions that estimate the distance-to-goal for any state and can be used for directing search. Abstraction is a popular method for deriving admissible heuristics, i.e., heuristics that never overestimate the true distances and that guarantee A* and IDA* to find optimal solutions. One creates an abstract version of the original state space and uses the true distances in the abstract state space as heuristic values, which are then stored in an efficient data structure called a pattern database (PDB) [5]. The PDB is essentially a table listing abstract states along with the corresponding heuristic values, and is

built by moving backwards starting from the abstract goal applying the abstract versions of the operators in a breadth-first manner. Unfortunately, PDBs 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 [25]. 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.

The purpose of this paper is to study the effects on search performance when errors are made by the mutex detection method (h^2 or MMM) that is informing the construction of a PDB. We will see that both methods are often perfect, i.e. they exactly identify which pairs are mutex and which are reachable. We show several cases in which one or the other is not perfect. When h^2 errs (fails to identify some of the mutexes), a moderately negative effect on search speed is observed, but the heuristics remain admissible. MMM's errors may introduce inadmissibility, but in our experiments, suboptimal solutions were rarely produced and, when they did occur, the suboptimality was always very small.

2 Background

In this section we introduce the methods behind h^2 and MMM. We further explain how PDB creation is modified when taking mutex pairs into account.

2.1 Mutex Detection Methods

Most existing mutex detection methods use invariant synthesis in the process of mutex detection. The state-of-the-art mutex detection method h^2 discovers mutex pairs as a special case of "at-most-one" invariants consisting of only two atoms.[3] The h^2 invariant synthesis process can be summarized as follows:

- The (pairs of) atoms of the initial state are reachable.
- An operator is considered applicable if all single atoms and pairs of atoms in its preconditions are reachable.
- An applicable operator turns reachable all its single add effects and all pairs made in one of the following ways:
  - from the add effects of the operators,
  - any add effect combined with any previous reachable atom which is not deleted by the operator and is not mutex with one of its preconditions.

MMM is a sampling-based method that can be summarized as follows [21]:

1. Fix a number N of (not necessarily distinct) pairs of variable-value assignments to be sampled.

[3] For more background on invariants the reader is referred to [12].

2. To sample at least N pairs of variable-value assignments, sample N_s states, where N_s is the smallest integer such that $N_s \cdot \binom{m}{2} \geq N$. All pairs of variable-value assignments are extracted from the N_s sampled reachable states.
3. If a pair of variable-value assignments is not seen in this process, it is considered to be a mutex pair.

The number N of pairs to be sampled is determined by Berend and Kontorovich's [1] upper bound on the expected probability mass of the elements not seen after taking N i.i.d. samples from any fixed distribution. A good sampling method of reachable states is also an essential part for the success of this approach. For details of the MMM approach see [21].

A 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. 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. Furthermore, even with uniform sampling there would be no *guarantee* that MMM finds all reachable pairs; however, we would have a guaranteed minimum probability of finding all reachable pairs. The potential for MMM to miss some reachable pairs must be taken into consideration when building PDBs.

2.2 Pattern Databases

A well-known approach for directing search is by using heuristic functions. These are functions that estimate the distance from any given state s to a goal state. A heuristic function is *admissible* if it never overestimate the actual distances. An admissible heuristic function guarantees that heuristic search algorithms like A* and IDA* find optimal solutions when using this function. By creating an abstract version of the original state space and using the true distances in this abstract state space, one can generate an admissible heuristic function. The speed up in search achieved by A* and IDA* depends on the quality of the heuristic function: the closer the heuristic values are to the actual distances, the more efficient A* and IDA* will be.

Creating the h^2-modified PDB is quite straightforward. The only difference to the original PDB creation is that while moving backwards from the abstract goal, an abstract state containing a mutex pair is not added to the open list.

Because MMM errs on the other side, the MMM-modified PDB creation needs more work. The corresponding steps can be summarized as follows:

1. The PDB is built as usual.
2. MMM is used forward from the start state in the original state space to find mutex pairs.

3. An auxiliary PDB is built finding paths to abstract states that do not pass through abstract states containing pairs that are considered mutex by MMM. This means that, in the process of creating this PDB, abstract states containing a mutex pair are not added to the open list.
4. Distances in the original PDB will be replaced by those from the auxiliary PDB as long as the latter are not infinite.

The above explanation is given this way for the ease of understanding the MMM-modified PDB creation process. The actual implementation of the described method is more efficient in the sense that instead of building a separate PDB, the states considered spurious are flagged in the original PDB. As usual, the final PDB will be used for guiding the search. Since MMM might flag some reachable pairs as mutex, the resulting heuristic might be inadmissible. Though this can cause A* or IDA* to find suboptimal solutions, the suboptimality is bounded (additively) by the maximum amount that MMM increases a value in the original PDB (p.219, [11]). It should be noted that a suboptimal path will be found only if *all* the optimal paths are blocked by an MMM mistake.

3 Experimental Setup

The purpose of our experiment is to evaluate the effects on search of h^2 failing to find all mutex pairs and MMM failing to find all reachable pairs. The choice of domains, representations, and abstractions were driven by this goal, and the results reported below are only for those combinations in which one or both of h^2 and MMM are "imperfect" in the sense just described. Such combinations are rather rare; for the majority of the combinations we considered both methods were perfect (this is consistent with the findings described in [21]). All domains were represented using PSVN [16]. The representations used for the domains were intentionally chosen so that many mutex pairs exist, and we deliberately chose versions of each domain that were small enough that (1) problem instances could be solved reasonably quickly with a single PDB; and (2) if we were not able to analytically compute the exact number of reachable pairs, the state space was small enough that we could determine the exact number of reachable pairs by enumerating all reachable states.

3.1 Domains and Representations

The experimented domains and representations are as follows:

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. The goal is to stack up all disks in decreasing order, from bottom to top, on the goal peg from a given start state using the legal moves.

We encode a state of the n-Disk Towers of Hanoi with p pegs as a vector of length pn, where for every peg n binary variables encode whether a disk is on

this peg or not. h^2 is perfect on the size of this domain that we tested (14 disks, 4 pegs), but MMM is not. This is different than the encoding used in [21].

Blocks World with Table Positions. In the n-Blocks World with p Table Positions, a state describes the constellation of n blocks stacked on a table with p named positions, where at most one block can be located in a "hand." In every move, either the empty hand picks up the top block off one of the stacks on the table, or the hand holding a block places that block onto an empty table position or on top of a stack of blocks. The goal is to stack up all numbered blocks in increasing order, from bottom to top, on the goal position from a given start state using the legal moves.

We consider two PSVN representations of the n-Blocks World with p distinct table positions. In the first one, called the *top representation* [21], a state vector has $1 + p + n$ components, each containing either the value 0 or one of n block names: (i) the first component is the name of the block in the hand, (ii) the next p components are the names of the blocks immediately on table positions 1 through p, (iii) the last n components are the names of the blocks immediately on top of blocks $a, b, c, \ldots$. In each case, the value 0 means "no block." In the versions of the Blocks World we considered (3 table positions, 9 and 12 blocks), h^2 and MMM are both perfect with this representation.

In the *height* representation, a state is a vector of length $1 + 3n + p$, where (i) the first component is the name of the block in the hand or 0 if the hand is free, (ii) for every block, 3 components encode its table position, its height relative to the table and whether there is any block on top of this block and (iii) the last p components encode whether there is any block on a table position. In the versions of the Blocks World we considered, MMM is perfect with this representation but h^2 is not.

Sliding-Tile Puzzle. In the $n \times m$-Sliding-Tile Puzzle ($n \times m$-puzzle for short), representing an $n \times m$ grid, in which tiles numbered 1 through $n \cdot m - 1$ each fill one grid position and the remaining grid position is blank. A move consists of swapping the blank with an adjacent tile. The goal is to have the numbered tiles in increasing order from top left corner to bottom right corner with the blank tile in the bottom right position.

In the *standard representation* of this puzzle, states are vectors of length $n \cdot \ell$, where each component corresponds to a grid position and represents 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 represents the grid position at which the corresponding tile is located. In the version of the Sliding-Tile puzzle we considered (3×4), h^2 and MMM are both perfect with both of these representations.

Scanalyzer. In the n-Belt Scanalyzer, 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 [15].) In a *rotate* move, a batch can be switched from one conveyor belt in the upper half 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 henceforward.

In the PSVN representation of the n-Belt Scanalyzer [15] (for even n), a state is 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. h^2 and MMM are both perfect on the size of this domain that we considered (12 belts), although prior research [21] shows that MMM is not perfect on sufficiently large sizes of this domain.

Barman. We use the multi-valued representation derived by Fast Downward's preprocessing algorithm from `barman-opt11-strips_prob01-003.pddl`. Neither h^2 nor MMM is perfect on this domain.

Transport. We use the multi-valued representation derived by Fast Downward's preprocessing algorithm from `transport-opt08-strips_prob01.pddl`. MMM is perfect on this domain but h^2 is not, although it is perfect on the larger versions we tried.

3.2 Mutex Calculation

In our experiments, MMM and h^2 each calculate their set of reachable pairs once for each domain, not once for each start state used for testing. All the domains except Barman and Transport have invertible operators. For these, the calculation of reachable pairs began at the goal state. For Transport and Barman this calculation began at the start state in the PDDL problem definition file.

3.3 MMM's Sampling Method

The sampling method used for MMM is a simple random walk modified to cope with non-invertible operators. While doing a random walk, when we generate a child state, if there is no operator that generates its parent when applied to the child, we add this child state, along with the parent, to a list of states that do not return to their parents. When generating a node in the random walk process, we check if a state belongs to this list. If it does, the parent of this state is also considered as a child in generating the next state of the random walk.[4] The length of the random walk was equal to the sample size N_s, as determined by the method described in [21] with $p = 0.00001$.

4 Experimental Results

Our experiments involve building a PDB based on the mutexes discovered by one of the methods (h^2 or MMM) and then solving a set of problem instances with IDA* using that PDB. For h^2, optimal solutions will be found so we measure

[4] Although this is not the most efficient approach for this purpose, it is enough for the purpose of this study.

search performance in terms of the number of nodes expanded compared to the number that would have been expanded if all mutex pairs had been known when the PDB was built. These results are presented in the first subsection below. For MMM, we measure the suboptimality of the solutions found in addition to the search performance. These results are presented in the second subsection below.

The problem instances (start states) used for evaluation are generated as follows. For Barman and Transport, the start states were chosen uniformly at random from the set of states reachable from the start state in the PDDL problem definition file (the state spaces were small enough that this set of reachable states could be enumerated). For the other domains, we generated start states uniformly at random using domain-specific knowledge.

4.1 Effect of Mutexes Missed by h^2

The smallest size of the Transport domain (the only size we tried on which h^2 is imperfect) has one state variable that can take on 5 values (0...4). We defined a domain abstraction that maps two of these values (0 and 4) to the same value (0). The average number of nodes expanded over 100 start states is 52 using the h^2-based PDB, compared to 46 for a PDB based on all mutex pairs. This difference represents a 13% reduction in search speed.

In the Barman representation there are 62 state variables, of which we projected out all except for variables 1, 4, 7, 10, 11, 15, 18, 19, 23, 27, 32, 36, 40, 41, 42, 43, 47, 50, 54, 58, and 62. If IDA* is run with the PDB based on this abstraction on our 100 start states, it expands $179,963,835$ nodes on average. The PDB based on the mutex pairs that h^2 finds reduces this only slightly, to $179,863,552$ nodes. When knowing all the mutex pairs, only $145,221,715$ nodes would be expanded, so the mutexes h^2 misses cause IDA* to be 24% slower.

In the height representation of the Blocks World with 9 blocks and 3 table positions we evaluate ten abstractions that project out 17-19 variables from the state representation. Table 1 compares the number of nodes expanded using the h^2-derived PDB with the PDB based on all mutex pairs. The numbers shown are an average over 1,000 problem instances with an average solution length of 37.073. The "Ratio" column shows that IDA* is between 7% and 17% slower because of the mutex pairs missed by h^2.

4.2 Effect of the Reachable Pairs Missed by MMM

In the Towers of Hanoi with 14 disks and 4 pegs there are $6,188$ reachable pairs and MMM found $5,783$ of them with the sample size it computed ($147,151$ states). The abstraction we used projected out 4 variables, the ones indicating whether or not disks 3, 7, 10, and 12 are on the goal peg (peg 4). Using the PDB based on this abstraction (which contained no spurious states) IDA* expanded $407,939,036$ nodes on average over 100 problem instances with an average solution length of 84.93. Using the MMM-based PDB gives slightly better results—always optimal solutions are found and $405,102,108$ nodes are expanded.

Table 1. Comparison of the h^2-based PDB with the PDB based on all mutex pairs for the Blocks World with 9 blocks and 3 table positions, in terms of the average number of nodes expanded by IDA*.

#	h^2 mutexes	all mutexes	Ratio
1	49,713,497	46,330,261	1.07
2	46,383,586	43,370,655	1.07
3	24,517,691	22,781,118	1.08
4	45,702,201	41,787,185	1.09
5	41,955,106	39,756,764	1.10
6	32,572,769	29,487,744	1.10
7	17,508,997	15,725,339	1.11
8	53,238,425	48,043,270	1.11
9	16,599,782	14,641,797	1.13
10	28,087,835	24,006,009	1.17

Table 2. Results on MMM-based PDBs for the Sliding-Tile puzzle, over various sample sizes used by MMM.

#Samples	# Pairs Found	Avg. Solution Length	# Nodes Expanded
5,000	7,743	34.987	8,260,410
7,500	8,306	35.381	4,034,902
10,000	8,679	34.871	4,721,158
12,500	8,798	34.743	4,900,564
15,000	8,829	34.737	4,873,844
20,000	8,851	34.737	4,876,682

In the Barman domain there are $8,546$ reachable pairs of which MMM found $7,834$ with the sample size it computed ($171,566$ states). The abstraction we used is the same as that used for h^2 (see above). Using the PDB based on this abstraction, IDA* expanded $179,963,835$ nodes on average over the same problem instances used for h^2. Using the MMM-based PDB gives exactly the same results—all solutions found are optimal and the same number of nodes is expanded. Using a PDB based on all mutex pairs only $145,221,715$ nodes would have been expanded.

To get additional data on the effect of MMM's mistakes we severely reduced the sample size it was given on two domains where the sample size it computed was sufficient to find all reachable pairs. The first such experiment was with the dual representation of the 3×4 Sliding-Tile Puzzle. The abstraction we used projects out tiles 1, 2, 6, 7, 8, 10 and 11. The optimum average solution length over 1,000 problem instances is 34.737. With the PDB for this abstraction IDA* expands $12,891,609$ nodes on average, and with the PDB based on all mutex pairs it expands $4,876,682$ nodes. There are $8,856$ reachable pairs in this domain. Table 2 shows, for various sample sizes, the number of reachable pairs MMM found in a sample of that size, and the average solution length and the number of nodes expanded using the MMM-based PDB for that sample size. For all sample sizes, the solutions are within 2% of optimal and the number of nodes expanded is close to the one using the PDB based on all mutex pairs.

The second such experiment was with the top representation of the Blocks World top with 12 blocks and 3 table positions. The abstraction we used mapped constant 0 and 1 to 0, constants 2, 3, and 4 to 1, constants 5 and 6 to 2, constants

7, 8, and 9 to 3, and constants 10, 11, and 12 to 4. The optimum average solution length over 100 problem instances is 50.02. With the PDB for this abstraction IDA* expands $338,837,610$ nodes on average, and with the PDB based on knowing all mutex pairs it expands $219,554,160$ nodes. There are $16,744$ reachable pairs in this domain. Table 3 shows, for a variety of sample sizes, the number of reachable pairs MMM found in a sample of that size, and the average solution length and the number of nodes expanded using the MMM-derived PDB for the given sample size. Even for small sample sizes, the solutions found are optimal and the number of nodes expanded is close to the number expanded using the PDB based on all mutex pairs. The average heuristic values over $1,000$ problem instances are also shown in this table. Although the inadmissibility of the heuristic due to missing reachable pairs can cause A* or IDA* to find suboptimal solutions (see the second row of the table), the suboptimality will be bounded (p.219, [11]).

Table 3. Results on MMM-based PDBs for the Blocks World with 12 blocks and 3 positions in top representation, over various sample sizes used by MMM. The average heuristic value of the original PDB over $1,000$ problem instances is 23.214.

#Samples	# Pairs Found	Avg. Solution Length	# Nodes Expanded	Average Heuristic
2,500	6,769	50.02	334,008,819	23.246
5,000	10,504	50.26	292,895,093	23.744
7,500	12,137	50.02	245,575,442	23.770
10,000	13,212	50.02	232,661,738	23.896
15,000	14,870	50.02	230,423,751	23.912
20,000	15,860	50.02	223,020,627	23.956
25,000	16,414	50.02	221,228,345	23.966
30,000	16,493	50.02	219,554,160	23.968
35,000	16,539	50.02	220,180,915	23.956
40,000	16,654	50.02	219,554,160	23.968
45,000	16,700	50.02	219,554,160	23.968
50,000	16,726	50.02	219,554,160	23.968

As a final example, consider a second abstraction of the Blocks World with 12 blocks and 3 table positions in top representation, one that contains no mutexes. The abstraction we used mapped constants 0, 1 and 2 to 0, constants 3, 4, and 5 to 1, constants 6, 7 and 8 to 2, constants 9, 10, 11, and 12 to 3. With the PDB for this abstraction IDA* expands $11,315,347,373$ nodes on average. There are $16,744$ reachable pairs in this domain. Table 4 shows, for a variety of sample sizes, the number of reachable pairs MMM found in a sample of that size, and the average solution length and the number of nodes expanded using the MMM-derived PDB for the given sample size. The average heuristic values over $1,000$ problem instances are also shown in this table. With every sample size, the solutions found are optimal. In some cases when the sample size is very small, the number of nodes expanded is less than the number expanded using the original PDB. This means that missing reachable pairs by MMM does not necessarily yield suboptimality; we can even gain a speed-up in search without sacrificing solution quality. As mentioned before, the inadmissibility caused by missing reachable pairs can make A* or IDA* find suboptimal solutions. How-

ever, the missing pairs do not cause any suboptimality in solution length in the 100 problem instances experimented here.

Table 4. Solution length, number of nodes expanded, averaged over 100 problem instances and average heuristic value of the MMM-based PDB over 1,000 problem instances of the Blocks World with 12 blocks and 3 table positions in top representation. The average heuristic value of the original PDB over 1,000 problem instances is 18.058.

#Samples	# Pairs Found	Avg. Solution Length	# Nodes Expanded	Average Heuristic
2,500	6,769	50.02	10,089,469,726	18.222
5,000	10,504	50.02	10,934,356,821	18.120
7,500	12,137	50.02	10,996,998,475	18.136
10,000	13,212	50.02	11,315,347,373	18.058
15,000	14,870	50.02	11,267,254,232	18.058
20,000	15,860	50.02	11,315,347,373	18.058
25,000	16,414	50.02	11,315,347,373	18.058
30,000	16,493	50.02	11,315,347,373	18.058
35,000	16,539	50.02	11,315,347,373	18.058
40,000	16,654	50.02	11,315,347,373	18.058
45,000	16,700	50.02	11,315,347,373	18.058
50,000	16,726	50.02	11,315,347,373	18.058

5 Conclusions

The purpose of this paper was to study the effects on search performance when errors are made by the mutex detection method that is informing the construction of a pattern database (PDB). We have considered two mutex detection methods, h^2, which can fail to recognize a mutex pair but never regards a reachable pair as mutex, and MMM, which makes the opposite type of error. Both methods are very often perfect, i.e., they exactly identify which pairs are mutex and which are reachable. In the cases that they err that we have examined in this paper, h^2's errors cause search to be moderately slower ($7\%-24\%$) than it otherwise would be, but its PDB is guaranteed to be an admissible heuristic. MMM's errors can cause its PDB to be an inadmissible heuristic but in our experiments its errors rarely caused a non-optimal solution to be found and when they did the solutions were extremely close to optimal. MMM's errors were expected to speed up search but this was not observed; they had little effect on search speed. Our experiments also showed that MMM can perform very well with quite small sample sizes.

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. and Geffner, H. Planning as heuristic search: New results. In: ECP, 360–372. Springer, 1999.
4. Chen, Y., Huang, R., Xing, Z., and Zhang, W. Long-distance mutual exclusion for planning. Artif. Intell., 173(2):365–391, 2009.
5. Culberson, J., Schaeffer, J.: Pattern databases. Comput. Intell. **14**(3) (1998) 318–334
6. Edelkamp, S., Helmert, M.: MIPS: The Model-Checking Integrated Planning System. AI Magazine **22**(3) (2001) 67–72
7. Fox, M. and Long, D. The automatic inference of state invariants in TIM. J. Artif. Intell. Res., 9:367–421, 1998.
8. Gerevini, A., Saetti, A., and Serina, I. Planning through stochastic local search and temporal action graphs in lpg. *J. Artif. Int. Res.*, 20:239–290, 2003.
9. Gerevini, A., Schubert, L.K.: Discovering state constraints in DISCOPLAN: Some new results. In: AAAI/IAAI. (2000) 761–767
10. Gerevini, A. and Schubert, L. Inferring state constraints for domain-independent planning. In AAAI/IAAI, pages 905–912, 1998.
11. Harris, L.R.: The heuristic search under conditions of error. Artificial Intelligence **5**(3) (1974) 217–234
12. Haslum, P.: Admissible Heuristics for Automated Planning. Linköping Studies in Science and Technology: Dissertations. Dept. of Computer and Information Science, Linköping Univ. (2006)
13. Haslum, P., Bonet, B., and Geffner, H. New admissible heuristics for domain-independent planning. In: AAAI, pages 1163–1168, 2005.
14. Helmert, M.: The Fast Downward planning system. J. Artif. Intell. Res. **26** (2006) 191–246
15. Helmert, M., Lasinger, H.: The Scanalyzer domain: Greenhouse logistics as a planning problem. In: ICAPS. (2010) 234–237
16. Hernádvölgyi, I., Holte, R.: PSVN: A vector representation for production systems. Technical Report TR-99-04, Dept. of Computer Science, Univ. of Ottawa (1999)
17. Kautz, H.: SATPLAN04: Planning as satisfiability. In: 4th International Planning Competition Booklet. (2004)
18. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search, AAAI Press (1996) 1194–1201
19. Penberthy, J. and Weld, D. Temporal planning with continuous change. In: AAAI, 1010–1015, 1994.
20. Rintanen, J. An iterative algorithm for synthesizing invariants. In: AAAI/IAAI, 806–811. 2000.
21. Sadeqi, M., Holte, R.C., Zilles, S.: Detecting mutex pairs in state spaces by sampling. In: 26th Australasian Joint Conference on Artificial Intelligence. (2013) 490–501
22. Sadeqi, M., Holte, R.C., Zilles, S.: Using coarse state space abstractions to detect mutex pairs. In: SARA. (2013) 104–111
23. Scholz, U. Extracting state constraints from PDDL-like planning domains. In: AIPS Workshop on Analyzing and Exploiting Domain Knowledge for Efficient Planning, 43–48, 2000.
24. Vidal, V. and Geffner, H. Branching and pruning: An optimal temporal pocl planner based on constraint programming. Artif. Intell., 170:298–335, 2006.
25. Zilles, S., Holte, R.C.: The computational complexity of avoiding spurious states in state space abstraction. Artif. Intell. **174** (2010) 1072–1092