Multiple Symmetries in Sliding-Tile Puzzles: First Experiments

César Romero Julio Castillo Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
{cesar,julio}@gia.usb.ve, bonet@ldc.usb.ve

Abstract

Since their introduction, symmetries have proven to be very powerful for the solution of different tasks related to heuristic search on sliding-tile puzzles. The most relevant being the boost of the heuristic values stored in Pattern Databases (PDBs), the construction time and storing size of PDBs, and the reduction in the number of written states in external search algorithms. Indeed, in the later, symmetries are almost a necessity in order to perform such complete search in acceptable time and space. All these applications only use single symmetries. As it turns out, multiple symmetries can also be utilized with very good results. In this paper, we explore the application of multiple symmetries on sliding puzzle problems and present some preliminary experimental results on complete explorations of state spaces using an external search algorithm. As it is shown, the net result is a 4-fold reduction in the number of written states with respect to an implementation that utilizes no symmetries.

Introduction

The first use of reflection symmetries in the sliding-tile puzzles seems to appear in Culberson and Schaeffer's seminal work on Pattern Databases (PDBs) (Culberson & Schaeffer 1998). In such work, symmetries are used to boost the heuristic value of a state s by taking the maximum of the PDB value for s and the PDB value for the reflection of s.

Since then, symmetries have proven to be very powerful for solving sliding-tile puzzles in one form or another. Symmetries have been used to boost the heuristic value of pattern databases (Korf & Felner 2002; Felner, Korf, & Hanan 2004; Felner *et al.* 2005), in a dual IDA* algorithm that switches between states and their reflections along paths dynamically (Zahavi *et al.* 2008), and to reduce the number of states written to disk in a complete exploration of the state space using an external search algorithm (Korf & Schultze 2005). The latter is a dramatic example of the potential savings reached by symmetries as their use permit to reduce the number of written states by half reducing the disk needs from near 3TB to 1.4TB.

Another application of symmetries, already noted in the work of Culberson and Schaeffer, is to reduce the computation time and size of large PDBs. Indeed, a single symmetry can reduce the size and computation time by almost a half yet special care must be taken in order to preserve compact ranking functions. Since PDBs have become the source of the best and most powerful heuristics for diverse problems (Korf 1997; Edelkamp 2001; Felner, Korf, & Hanan 2004; Holte *et al.* 2006), the effective use of symmetries then becomes a fundamental tool.

All these applications always utilize one symmetry of the given puzzle among several candidates. However, given the success stories just described, we asked whether it was possible to utilize more than one symmetry simultaneously and, in such case, what would be the benefits.

It turns out that the answer is affirmative, and our first findings show that the benefits are quite good. Indeed, the use of one symmetry reduces in half the number of written states in an external search algorithm, yet the use of two symmetries simultaneously reduces the latter by a factor of 2 for a total reduction by a factor of 4.

In this paper, we describe the use of multiple symmetries in three fundamental tasks related to the sliding-tile puzzles: the boost of heuristic values given by PDBs, the reduction in time and size for the construction of PDBs, and the reduction in the number of written states in complete explorations of state spaces using external search algorithms. Besides these contributions, the paper offers a comprehensive exposition of symmetries that we also consider part of the contribution.

The paper is organized as follows. In the next section, we offer a complete exposition of permutations and symmetries in the puzzle, and derive the equations that relate reflected paths and states with respect to their unreflected counterparts. Then, we show how symmetries can be exploited in the three fundamental tasks, and explain how the multiple symmetries can be used concurrently instead of a single one. Some preliminary experiments for the case of complete explorations of state spaces using external search are presented. The paper finishes with discussions of future work and conclusions.

Permutations and Symmetries

We use permutations to define the different elements that compose the sliding-tile puzzles, and then utilize them to define symmetries on the puzzle, paths and symmetric paths, and the relations among such concepts.

We assume the reader is knowledgeable with the sliding-tile puzzles and that has some elementary knowledge of permutations. In this paper, the composition of two permu-

tations s and t is denoted by multiplication $s \cdot t$, or even st, and corresponds to the permutation $(st)(i) = t(s(i))$. All permutations will be defined over sets of the form $\{0, \ldots, n-1\}$ denoted by $[n]$. The set of permutations over $[n]$ with composition form a multiplicative group; in particular, the composition operation is associative and each permutation s has an inverse s^{-1}.

Everything is a Permutation

In our view of the puzzle, everything constitutes a permutation: states (configurations) of the puzzle and operators. If we are dealing with a $n = rows \times cols$ puzzle, all permutations are over $[n]$. If s denotes a configuration, then $s(i)$ is the tile at position i with 0 denoting the blank.[1]

For simplicity, let us assume for the moment a 3×3 puzzle. In order to represent the operators as permutations, we qualify each operator with the blank position. Thus, for example, for the 'up' operator that moves the blank one position up in the puzzle, there are 6 operators $\mathsf{up}_3, \mathsf{up}_4, \ldots, \mathsf{up}_8$ such that up_i refers to the operator when the blank is in position i. As the up operator cannot be applied when the blank is in the first row, there is no up_i operator for $i = 0, 1, 2$.

Each operator can be understood as a permutation. For example, the operator up_5, that moves the blank up when it is located in position 5, is the permutation

$$\mathsf{up}_5 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 1 & \mathbf{5} & 3 & 4 & \mathbf{2} & 6 & 7 & 8 \end{pmatrix}$$

since if s is a configuration of the puzzle with the blank at position 5, then $\mathsf{up}_5 \cdot s$ is the configuration that results after moving the blank up one position. Observe that two operators can refer to the same permutation, e.g. $\mathsf{up}_5 = \mathsf{down}_2$ and in general every operator equals its inverse operator.

If we apply a sequence of movements, say $a_1, a_2, \ldots, a_m$ to state s, then the resulting state can be expressed as

$$(a_m \cdots (a_2(a_1 s)) \cdots) = (a_m \cdots a_1)s = \pi s$$

where π is the *macro-operator* (permutation) that results of applying all operators in the given order, i.e. $\pi = a_m \cdots a_1$. We call from now on, such macro-operators as paths as they correspond to the different paths in the search tree generated from a given initial configuration s_0. A sequence of operators applicable at s is called a valid sequence or path at s. If π is a valid path at s_0 that leads to state $s = \pi s_0$, then we say that s is reached by π.

Symmetries

A symmetry D of the puzzle is a permutation of the tiles in the puzzle, i.e. a permutation over $[n]$.

The intuitive idea behind the use of symmetries is the following. A symmetry D induces a mapping (automorphism) over the paths in the search tree rooted at the initial configuration s_0 in which the image of a path π is denoted by π^D. If s belongs to the search tree, then there is a path π that induces s, and so we can talk about the state s^D that is reached by π^D. If π^D has the same length as π, then s and s^D are at the same depth in the tree, and furthermore, if s^D can be computed efficiently from s, then we can manage to prune s^D from the search as no information is loss.

A formalization of above intuition must meet three requirements:

R1. if π is a path in the search tree, then π^D is also a path in the search tree, and vice versa,

R2. the length of π^D equals the length of π, and

R3. s^D and s must be easily computable from each other. In particular, there should be no need to keep track of either path π or π^D explicitly.

Let s be a state in the search tree and π its inducing path. In order to define π^D, we ask for the following property

$$Ds = D(\pi s_0) = (D\pi)s_0 = (\pi^D D)s_0 = \pi^D(Ds_0)$$

where the third equality is the property we require. In words, that the symmetric image of s should be equal to the state reached by π^D on the symmetric image of s_0.

This equation implies $D\pi = \pi^D D$ and thus $\pi^D = D\pi D^{-1}$. Therefore, if $\pi = a_1 a_2 \cdots a_n$, then

$$\begin{aligned} \pi^D &= Da_1 a_2 \cdots a_n D^{-1} \\ &= (Da_1 D^{-1})(Da_2 D^{-1}) \cdots (Da_n D^{-1}) \\ &= a_1^D a_2^D \cdots a_n^D \end{aligned}$$

where $a_i^D \doteq Da_i D^{-1}$ is defined as the *symmetric operator* of a_i. Henceforth, requirement R2 is clearly satisfied. The path π^D is also called the symmetric or reflected path.

For example, let D be a *reflection* of the puzzle along its main diagonal

$$D = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 3 & 6 & 1 & 4 & 7 & 2 & 5 & 8 \end{pmatrix}.$$

Under D, the operators right, down, left and up become the operators down, right, up and left respectively; e.g. $\mathsf{up}_5^D = \mathsf{left}_7$, $\mathsf{right}_7^D = \mathsf{down}_5$, $\mathsf{down}_0^D = \mathsf{right}_0$, etc.

Once reflected paths are defined, requirement R1 simply becomes a property of the symmetry with respect to the initial configuration:

Definition 1 (Admissible Symmetry) *A symmetry D is* admissible *with respect to an initial configuration s_0 if and only if for every* valid *path π at s_0, π^D is also* valid *at s_0.*

The only admissible symmetries for the sliding-tile puzzles are reflections along its main diagonals for square puzzles, or horizontal reflections along the middle column for puzzles with odd number of columns, or vertical reflections along the middle row for puzzles with odd number of rows. Furthermore, in each case, the blank in the initial configuration must be on the reflection axis since, to achieve admissibility, the blank must not be moved by the reflection. For any such symmetry D, we have $D = D^{-1}$ and thus $(a^D)^D = a$ and $a^D = DaD$ for all operators a.

Finally, for the last requirement, if s is reached by π, then to compute s^D (that is equal to $\pi^D s_0$ by definition) consider an expression of the form $s^D = DsE$:

$$\pi^D s_0 = DsE = D(\pi s_0)E = \pi^D(Ds_0E).$$

[1] An alternative representation is that $s(i)$ denotes the position of tile i. Both representation are equivalent yet the mathematical formulation changes a bit.

We need $s_0 = Ds_0E$ and thus E must be $s_0^{-1}D^{-1}s_0$. Then, s^D can be computed in constant time, in the length of π, as DsE. Similarly, for $F = s_0^{-1}Ds_0$, $s = D^{-1}s^DF$.

Exploiting Symmetries

There seem to be three major application of symmetries in sliding-tile puzzles. First, they are used to improve the heuristic values that arise from pattern databases (Culberson & Schaeffer 1998). Second, they can be used to reduce the construction time and storage of large PDBs. (This is application is mentioned in (Culberson & Schaeffer 1998) but we have not found further references of it, even in papers that deal with the compression of PDBs (Felner *et al.* 2004).) Third, they are used to reduce the number of states written to disk during a complete exploration of the puzzle's state space with an external search algorithm (Korf & Schultze 2005). In the last application, the number of written states is reduced in half and thus the space requirements. Also, the overall search time is also reduced in about half since the I/O operations are the most critical during an external search.

In this section, we explain how to use symmetries in above three applications.

Pattern Databases

The standard application of symmetries with PDBs consists to improve the heuristic value $h(s) = PDB[s]$ as $h'(s) = \max\{PDB[s], PDB[s^D]\}$, where $PDB[s]$ refers to the value stored for s in the pattern database. If the PDB is admissible, then h' is also admissible since s and s^D are at the same depth in the search tree and thus at the same distance from the goal. This boosting have shown to be very effective in (Culberson & Schaeffer 1998; Korf & Felner 2002; Felner, Korf, & Hanan 2004). Another type of boosting also based on symmetries and called dual lookups is defined in (Felner *et al.* 2005).

We can also apply symmetries to reduce the construction time of PDBs. Typically, a PDB is computed by a retrograde analysis with a breadth-first search that start at the goal pattern and spans the whole pattern space. The symmetries behave similarly with states and patterns.

For example, consider an abstraction that replaces tiles 5, 6, 7 and 8 by a single constant 'x'. If

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ x & 1 & x & 2 & 0 & x & 3 & x & 4 \end{pmatrix}$$

is a pattern, then

$$\mathsf{up}_4 \cdot p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ x & 0 & x & 2 & 1 & x & 3 & x & 4 \end{pmatrix}$$

is the pattern that results of moving the blank up.

The PDB can be constructed more efficiently with a diagonal symmetry D using a breadth-first search algorithm seeded at the goal pattern

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 1 & 2 & 3 & 4 & x & x & x & x \end{pmatrix},$$

and that *does not explore patterns that have the blank in the upper half of the puzzle* (as defined by D); that is, that does not insert such patterns into the queue. Then, whenever a pattern p is picked from the queue at depth d, not only the value $PDB[p] = d$ is set but also the value $PDB[p^D] = d$.

There is an important detail that must explained when making such construction. Since every pattern p typically represents two pattern in the pattern space, i.e. p and p^D, special care is needed when the blank is on the diagonal (the symmetry axis). In this case, p still can represent two different patterns both with the blank in the same position, and thus when expanding such patterns, the reflection p^D should also be computed and expanded during the search.

This technique may reduce in half the construction time of the pattern database. As also noted in (Culberson & Schaeffer 1998), we can even use symmetries to reduce the size of the PDB by storing only the entries for p and not for p^D. This however needs special handling to permit compact and efficient indexing into the PDB.

External Search

We explain how to use symmetries within an external frontier search that avoid the re-generation of interior nodes. Since the search space for the puzzle is undirected, i.e. each operator has an inverse, it is enough to associate with each state a list of forbidden operators. Thus, every time that state s is reached from state s' through operator a, then the inverse operator a^{-1} is inserted into the forbidden list for s. At expansion time, only operators that do not appear in the forbidden list for s are applied at s.

An external frontier search works by processing the nodes in the search tree layer by layer and never storing the interior nodes of the tree. A typical iteration consists of an expansion phase followed by a duplicate detection phase. The expansion of all nodes in the kth layer generates the node at the $(k+1)$th layer, duplicates included. Then, such duplicates at the $(k+1)$th layer are removed either using sort-based duplicate detection (Korf 2003), hash-based duplicate detection (Korf 2004) or other (Zhou & Hansen 2004). Furthermore, each non-duplicate node in the $(k+1)$th layer is accounted for in the statistics. At the end of the duplicate detection, the nodes at layer k are deleted, and the algorithm moves on to the expansion phase for the nodes at layer $k+1$.

When using a single symmetry D, each node s encountered during the search represents at most two states $\{s, s^D\}$. This set is called the class of s modulo D and is denoted by $[s]_D$. Since $D = D^{-1}$, we have that for all states s, s', either $[s]_D = [s']_D$ or $[s]_D \cap [s']_D = \emptyset$, and this is the fundamental property that permit the successful exploitation of symmetries. Otherwise, double counting of states may occur if this property does not hold.

The next step is to designate one and only one state in $[s]_D$ as the class representative. For example, with the diagonal symmetry D, we have that if s has the blank above the diagonal, then s^D has the blank below the diagonal. In this case, states with the blank below the diagonal can be designated as representatives. However, if s has the blank in the diagonal, then s^D also has it in the diagonal. In this case, the representative must be designated in another way, e.g. lexicographically.

The idea is to substitute each state encountered during the search by its representative. Thus, during expansion, each generated node is replaced by its representative, and during duplicate detection, each non-duplicate is accounted for the number of nodes it represents, either 1 or 2 whether $s = s^D$. Moreover, during expansion, if s is reached from s' through operator a, then operator a^{-1} is insert into the forbidden list as well as $(a^D)^{-1}$ if $s = s^D$.

Multiple Symmetries

A natural extension of above results is obtained by using multiple symmetries instead of a single symmetry.

Let us consider the set $G = \{D, A\}$ of the diagonal symmetries for the 3×3 puzzle; the reflection D along the main diagonal and the reflection A along the main antidiagonal:

$$D = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 3 & 6 & 1 & 4 & 7 & 2 & 5 & 8 \end{pmatrix},$$
$$A = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 5 & 2 & 7 & 4 & 1 & 6 & 3 & 0 \end{pmatrix}.$$

This time, for each state s, the class of s modulo G is the set $\{s, s^D, s^A, s^{AD}, s^{DA}\}$ which coincides with $\{s, s^D, s^A, s^{AD}\}$ as the permutations A and D commute. As before, it is not difficult to show that for all states s, s', then either $[s]_G = [s']_G$ or $[s]_G \cap [s']_G = \emptyset$.

In order to use both symmetries concurrently, it is required that the initial state (or goal state in case of PDBs) to be admissible with respect to both symmetries simultaneously, i.e. that the blank be in position 4. So, let us assume that in this case the initial state is

$$s_0 = s_{3\times 3} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 1 & 2 & 3 & 0 & 5 & 6 & 7 & 8 \end{pmatrix}.$$

We show how an external search algorithm (and similarly the construction of a PDB) only needs to store states in which the blank is in position 4, 6, 7 or 8, and can safely discard all other states. For each state s, define the representative of $[s]_G$ as a fixed state for which the blank is either in position 4, 6, 7 or 8. Finding representatives for states that have the blank in $\{0, 1, 2, 3, 5, 7\}$ is easy as it can be done with the application of a permutation, yet if the blank is in $\{4, 6, 8\}$, then some of their images can have the blank also in $\{4, 6, 8\}$ and the computation might require lexicographic ordering. Figure 1 shows both diagonals D and A, and the permutations needed to compute the representative when the blank's position is in $\{0, 1, 2, 3, 5, 7\}$.

During the expansion phase, each state s is substituted by its representative, and some of the operators are inserted into the forbidden list for s: a^{-1} is always inserted, and $(a^D)^{-1}$, $(a^A)^{-1}$ and $(a^{AD})^{-1}$ are inserted whether $s = s^D$, $s = s^A$ and $s = s^{AD}$ (some of these may happen simultaneously). The last three cases are only possible when the blank is over some symmetry axis.

During duplicate detection, each non-duplicate is accounted for the correct number of times. In this case, each such state may represent either 1, 2 or 4 other states as given in Table 1. The case of a unit count is interesting as this is only possible when the blank is in the middle and all images

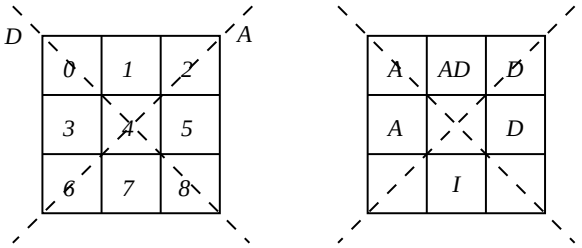

Figure 1: The diagonal and antidiagonal reflections in the 3×3 puzzle. Only states that have the blank in positions 4, 6, 7 or 8 need to be written to disk in a complete exploration of the state space. The right panel shows the symmetries that need to be applied to transform a given state into one with the blank in $\{4, 6, 7, 8\}$.

condition	count
$s = s^D, s = s^A$	1
$s = s^D, s \neq s^A$	2
$s \neq s^D, s = s^A$	2
$s \neq s^D, s \neq s^A, s^A = s^D$	2
$s \neq s^D, s \neq s^A, s^A \neq s^D$	4

Table 1: Different possible counts for each state s encountered during a complete exploration of the state space starting with the blank at the middle and using two reflections D and A.

are identical. In the 3×3 puzzle with the D and A symmetries, there are only 7 such states (not counting the initial state): 3 states at depth 24 and 4 states, in the last layer, at depth 30. The following is one such state at depth 24

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 5 & 2 & 7 & 0 & 1 & 6 & 3 & 8 \end{pmatrix}.$$

We can also consider other sets of reflections. For example, the set $G = \{H, V\}$ for the horizontal and vertical reflections

$$H = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 1 & 0 & 5 & 4 & 3 & 8 & 7 & 6 \end{pmatrix},$$
$$V = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 6 & 7 & 8 & 3 & 4 & 5 & 0 & 1 & 2 \end{pmatrix}.$$

This case is very similar to the case of the diagonal reflections $\{A, D\}$. A more interesting case arises with $G = \{D, H\}$ as these symmetries *do not commute* (see Fig. 2. Therefore, in order to guarantee the fundamental property, we need to consider elements other than $\{s, s^D, s^H, s^{DH}\}$. Indeed, one needs to consider the eight elements $\{s, s^D, s^H, s^{HD}, s^{DH}, s^{DHD}, s^{HDH}, s^{DHDH}\}$. Nonetheless, once we have defined the classes modulo G, the procedure is the same. During expansion, every time a state s is generated, s is replaced by a fixed representative chosen from $[s]_G$ and the forbidden list is updated. Then, during duplicate detection, each non-duplicate is accounted for a fixed number of times.

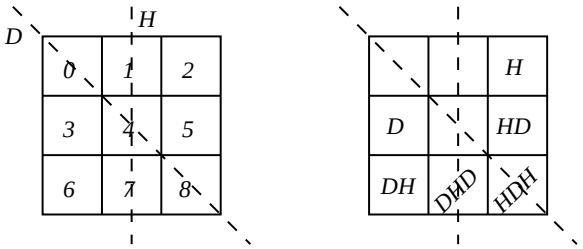

Figure 2: The diagonal and horizontal reflections in the 3×3 puzzle. Only states that have the blank in positions 0, 1 or 4 need to be written to disk in a complete exploration of the state space. The right panel shows the symmetries that to be applied to transform a given state into one with the blank in $\{0, 1, 4\}$.

Above discussion applies either to an external search algorithm or the construction of PDBs. In the latter case, each time a state is removed from the queue, not only it but all the states in its class modulo G, obtain a value in the PDB. Likewise, only representative states are inserted into the queue.

It is important to remark that a group of symmetries can only be applied to a search tree that is rooted in a state for which all symmetries are admissible. This is the reason for placing the blank at the middle for the groups $\{D, A\}$, $\{H, V\}$ and $\{D, H\}$. Observe that in the 4×4 puzzle, there is no state that makes more than one symmetry admissible since there is no middle position. Thus, at the moment, we do not know how to exploit multiple symmetries in the 4×4 puzzle.

The 5×5 puzzle is different as there is a middle position and also the horizontal and vertical symmetries are admissible. Although, at the moment, it seems impossible to make a complete exploration of the state space (even with multiple symmetries), multiple symmetries can be used to reduce the space needed to store PDBs for the 5×5 puzzle.

Preliminary Experiments

We have performed some preliminary experiments with multiple symmetries. In a first experiment, we performed a complete exploration of the 3×3 puzzle with the group of symmetries $\{D, A\}$ starting at the state $s_{3\times3}$ that has the blank in the middle.

The results are shown in Table 2 that contains, per layer in the search, the number of states in the layer, the number of sterile states (i.e. those that have no children as their forbidden lists are full, and thus they do not need to be written to disk), and the number of written states. Additionally, the table presents the distribution of counts for every layer in the tree. For example, at depth 18, there are $11,132$ states from which 196 are sterile. The search algorithm generates at this layer, $2,795$ representatives since $2,771$ of them represent 4 states and the remaining 24 represent 2 states for the total $11,132 = 2,599 \times 4 + 24 \times 2$. In the table, all rows satisfy

$$\#states = 4 \times (\#\times 4) + 2 \times (\#\times 2) + (\#\times 1)\,,$$
$$\#sterile + \#written = (\#\times 4) + (\#\times 2) + (\#\times 1)\,.$$

The number of representatives that represent 4 states each makes up 99.74% of the total number of states. Observe the ratio of 21.70% of written nodes. The same complete exploration without the use of symmetries needs to write 86.10% of states (recall that sterile states are not written), and the exploration with one symmetry needs to write 43.41% of the total number of states. Thus, the addition of an extra symmetry reduces the latter number by 2. We also performed experiments with the symmetries $\{V, H\}$ obtaining results similar to Table 2.

A much larger experiment was conducted for the 3×5 puzzle, the largest before the 4×4 puzzle in which we can apply multiple symmetries. In this case, since the puzzle is rectangular, we applied vertical and horizontal symmetries $\{V, H\}$ starting with a state with the blank at the middle. The results are shown in Table 3 (at the end of the paper) with the same format as Table 2. This time the number of representatives that represent 4 states each is 99.99% of the total number of states. Again, the number of nodes written to disk is only 21.89% of all states that means around 37GB of space in the largest layer. The exploration with one symmetry writes 43.79% of states and requires around 75GB of space for the largest layer, while the exploration without symmetries writes 87.5% of the states.

Future Work

Our results show that the use of multiple symmetries can have a profound impact on the performance of mainstream techniques in heuristic search, at least for the sliding-tile puzzles. Far from offering a definite answer on the role of symmetries, we believe that this work raises more questions than the answers it provides.

Among such questions, we are specially interested in the following. First, there is the issue of using non-commutative groups of symmetries, e.g. the group $\{D, H\}$ in the 3×3 puzzle shown in Fig. 2, which seems to have a great potential to reduce the number of explored states. Furthermore, there is also the question of using more than two symmetries on the puzzle. These two issues are closely related.

Second, as mentioned earlier, we cannot directly use two symmetries over the 4×4 puzzle as there is no position for the blank that makes both symmetries admissible. Therefore, we are left with the question of how to exploit two symmetries in this puzzle. This question is closely related to the question of how to relax the admissibility requirement on symmetries. Among other things, we think that the puzzle might be embedded or mapped into another instance for which admissible symmetries exist.

A third open question is whether multiple symmetries can also be applied to other permutation games. A very attractive domain is a variation of sliding-tile puzzles in which the first row is "connected" with the last row, and the leftmost column is also "connected" with the rightmost column, making up a torus-shaped puzzle. In such game, all permutations become admissible, since there are no restrictions on the operators, and thus we can apply any number of symmetries concurrently.

Another permutation game that seems attractive is Rubik's Cube. In the cube, all operators are applicable at every

d	states	sterile	written	$\times 1$	$\times 2$	$\times 4$
0	1	0	1	1	0	0
1	4	0	1	0	0	1
2	8	0	2	0	0	2
3	8	0	2	0	0	2
4	16	0	4	0	0	4
5	32	0	8	0	0	8
6	60	2	14	0	2	14
7	72	0	18	0	0	18
8	136	0	36	0	4	32
9	200	0	50	0	0	50
10	376	4	90	0	0	94
11	512	0	128	0	0	128
12	964	6	236	0	2	240
13	1,296	0	324	0	0	324
14	2,368	18	576	0	4	590
15	3,084	0	771	0	0	771
16	5,482	51	1,323	0	7	1,367
17	6,736	4	1,680	0	0	1,684
18	11,132	196	2,599	0	24	2,771
19	12,208	33	3,019	0	0	3,052
20	18,612	482	4,179	0	16	4,645
21	18,444	112	4,499	0	0	4,611
22	24,968	1,107	5,149	0	28	6,228
23	19,632	327	4,581	0	0	4,908
24	22,289	1,432	4,174	3	63	5,540
25	13,600	385	3,015	0	0	3,400
26	11,842	1,206	1,773	0	37	2,942
27	4,340	272	813	0	0	1,085
28	2,398	377	238	0	31	584
29	472	30	88	0	0	118
30	148	45	0	4	10	31
	181,440	6,089	39,390	8	228	45,244

Table 2: Complete exploration of the 3×3 puzzle's space starting at the state $s_{3\times 3}$, with the blank in the middle, and with the two diagonal symmetries $\{A, D\}$. The table contains, per level, the number of total, sterile and written states, and the distributions of how many states are represented per each state. The ratio (#written/#states) is 21.70%.

configuration and thus every possible symmetry is admissible. It seems that we can designate one corner of the cube as the marked corner and then consider three orthogonal diagonal symmetries along each dimension so that only states that have the marked corner "above" the diagonal should be explored. In this case, it would be also interesting to try non-commutative symmetries in the cube.

Conclusions

We have revisited the role of symmetries in the sliding-tile puzzles with emphasis on their usage with PDBs and external search. In our exposition, we formally derived the equations that are commonly used with symmetries and showed how and when multiple symmetries can be used concurrently in some puzzles. Although the use of (single) symmetries has been around for some time, we were not able to find a formal derivation of the equations that rule their usage.

Our first experiments showed that by using two symmetries in an external search of the 3×3 and 3×5 puzzle, the total number of states written to disk decreased to roughly 22% of the total thus improving by a factor of 2 the previous approach that uses a single symmetry. We believe that the use of non-commutative symmetries and/or the use of more symmetries can reduce this number even more.

Several main questions remain open such as to how to use non-admissible symmetries, how to use symmetries in other permutation games, and others.

References

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

Edelkamp, S. 2001. Planning with pattern databases. *Proc. 6th European Conf. on Planning*, 13–24.

Felner, A.; Meshulam, R.; Holte, R. C.; and Korf, R. E. 2004. Compressing pattern databases. *Proc. 19th National Conf. on Artificial Intelligence*, 638–641.

Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R. C. 2005. Dual lookups in pattern databases. *Proc. 19th International Joint Conf. on Artificial Intelligence*, 103–108.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research* 22:279–318.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170:1123–1136.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134:9–22.

Korf, R. E., and Schultze, P. 2005. Large-scale, parallel breadth-first search. *Proc. 20th National Conf. on Artificial Intelligence*, 1380–1385.

Korf, R. E. 1997. Finding optimal solutions to rubik's cube using pattern databases. *Proc. 14th National Conf. on Artificial Intelligence*, 700–705.

Korf, R. E. 2003. Delayed duplicate detection: Extended abstract. *Proc. 18th International Joint Conf. on Artificial Intelligence*, 1539–1541.

Korf, R. E. 2004. Best-first frontier search with delayed duplicate detection. *Proc. 19th National Conf. on Artificial Intelligence*, 650–657.

Zahavi, U.; Felner, A.; Holte, R.; and Schaeffer, J. 2008. Duality in permutation space and the dual search algorithm. *Artificial Intelligence* 172(4–5):514–540.

Zhou, R., and Hansen, E. 2004. Structured duplicate detection in external-memory graph search. *Proc. 19th National Conf. on Artificial Intelligence*, 683–688.

d	states	sterile	written	$\times 1$	$\times 2$	$\times 4$
0	1	0	1	1	0	0
1	4	0	2	0	2	0
2	10	0	3	0	1	2
3	20	0	5	0	0	5
4	32	0	8	0	0	8
5	72	0	18	0	0	18
6	144	0	36	0	0	36
7	304	1	75	0	0	76
8	548	0	138	0	2	136
9	1,160	1	289	0	0	290
10	2,108	0	527	0	0	527
11	4,336	10	1,074	0	0	1,084
12	7,568	0	1,895	0	6	1,889
13	15,456	39	3,828	0	6	3,861
14	27,244	4	6,809	0	4	6,809
15	54,748	170	13,519	0	4	13,685
16	93,984	20	23,487	0	22	23,485
17	186,832	587	46,127	0	12	46,702
18	317,026	115	79,158	0	33	79,240
19	619,784	2,395	152,566	0	30	154,931
20	1,029,048	580	256,719	0	74	257,225
21	1,978,016	8,779	485,745	0	40	494,484
22	3,215,064	2,351	801,464	0	98	803,717
23	6,063,064	31,807	1,483,988	0	58	1,515,737
24	9,626,543	9,020	2,397,719	3	202	2,406,534
25	17,798,478	108,695	4,340,984	0	119	4,449,560
26	27,552,286	33,718	6,854,511	2	312	6,887,915
27	49,826,898	358,321	12,098,483	0	159	12,456,645
28	75,028,026	118,148	18,639,112	2	504	18,756,754
29	132,527,350	1,116,718	32,015,244	0	249	33,131,713
30	193,676,616	393,493	48,026,130	2	935	48,418,686
31	333,410,942	3,299,918	80,053,035	0	435	83,352,518
32	471,843,720	1,220,078	116,741,606	2	1,505	117,960,177
33	790,021,118	9,202,202	188,303,428	0	701	197,504,929
34	1,079,752,930	3,547,695	266,391,792	4	2,503	269,936,980
35	1,753,248,266	24,093,048	414,219,597	0	1,157	438,311,488
36	2,305,427,741	9,623,979	566,734,785	7	3,647	576,355,110
37	3,618,206,568	58,718,411	845,833,996	0	1,530	904,550,877
38	4,558,643,762	24,050,358	1,115,613,487	2	5,806	1,139,658,037
39	6,890,001,516	131,896,668	1,590,604,857	0	2,292	1,722,499,233
40	8,280,693,048	54,852,287	2,015,324,934	6	7,909	2,070,169,306
41	12,013,997,414	269,509,279	2,733,991,680	0	3,211	3,003,497,748
42	13,723,004,388	112,500,822	3,318,255,840	4	11,124	3,430,745,534
43	19,070,493,398	495,582,240	4,272,043,164	0	4,109	4,767,621,295
44	20,653,531,394	205,844,319	4,957,545,295	16	13,507	5,163,376,091
45	27,459,536,790	815,584,838	6,049,301,983	0	5,247	6,864,881,574
46	28,162,697,917	334,918,175	6,705,765,068	9	17,514	7,040,665,720
47	35,811,545,160	1,199,089,520	7,753,800,039	0	6,538	8,952,883,021
48	34,766,871,474	485,116,342	8,206,611,650	24	20,211	8,691,707,757
49	42,278,061,352	1,577,446,491	8,992,072,660	0	7,626	10,569,511,525
50	38,833,813,137	627,907,289	9,080,557,152	13	22,294	9,708,442,134
51	45,136,428,198	1,860,677,113	9,423,434,085	0	8,297	11,284,102,901
52	39,185,445,148	728,534,556	9,067,838,565	18	23,641	9,796,349,462
53	43,485,466,670	1,968,710,216	8,902,660,729	0	8,555	10,871,362,390
54	35,625,894,156	758,308,713	8,148,176,566	18	23,453	8,906,461,808
55	37,680,035,252	1,867,284,351	7,552,728,573	0	8,222	9,420,004,702
56	29,061,983,265	706,769,763	6,558,737,071	11	22,019	7,265,484,804
57	29,218,691,400	1,583,534,032	5,721,142,710	0	7,784	7,304,668,958

d	states	sterile	written	$\times 1$	$\times 2$	$\times 4$
58	21,141,855,595	587,271,682	4,698,202,055	9	19,663	5,285,454,065
59	20,131,181,830	1,194,423,082	3,838,375,819	0	6,887	5,032,792,014
60	13,595,850,697	432,097,571	2,966,873,499	17	16,766	3,398,954,287
61	12,197,319,766	794,647,715	2,254,685,071	0	5,689	3,049,327,097
62	7,632,079,418	278,717,293	1,629,309,341	14	13,538	1,908,013,082
63	6,401,776,150	460,656,376	1,139,789,920	0	4,517	1,600,441,779
64	3,668,776,766	155,339,254	761,859,941	14	9,986	917,189,195
65	2,844,069,196	228,033,432	482,985,551	0	3,368	711,015,615
66	1,467,539,543	73,048,506	293,839,873	13	6,967	366,881,399
67	1,033,405,322	93,457,540	164,894,913	0	2,245	258,350,208
68	467,731,154	27,953,665	88,981,390	16	4,509	116,930,530
69	291,118,998	30,218,810	42,561,735	0	1,591	72,778,954
70	110,894,118	8,197,202	19,527,575	12	2,477	27,722,288
71	58,368,580	7,111,516	7,481,064	0	870	14,591,710
72	17,504,309	1,660,764	2,715,934	7	1,231	4,375,460
73	7,239,640	1,057,758	752,357	0	410	1,809,705
74	1,544,203	192,720	193,566	3	466	385,817
75	452,388	77,709	35,466	0	156	113,019
76	63,732	10,186	5,798	4	96	15,884
77	12,138	2,242	810	0	35	3,017
78	1,285	228	108	1	28	307
79	202	24	30	0	7	47
80	41	4	8	1	2	9
81	20	0	5	0	0	5
82	10	0	3	0	1	2
83	4	0	2	0	2	0
84	1	1	0	1	0	0
	653,837,184,000	20,294,182,955	143,165,285,845	256	345,216	163,459,123,328

Table 3: Complete exploration of the 3×5 puzzle's space starting at the state $s_{3\times5}$, with the blank in the middle, and with horizontal and vertical symmetries. The table contains, per level, the number of total, sterile and written states, and the distributions of how many states are represented per each state. The ratio (#written/#states) is 21.89%.