On the Hardness of 6x6 Rush Hour

An exploration of the entire configuration space

Author

Jelle van Assema

10200363

Supervisor

Daan van den Berg

June 26, 2014

Afstudeerproject Thesis

Bachelor Kunstmatige Intelligentie

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Universiteit van Amsterdam

Abstract

6x6 Rush Hour is a sliding block game where blocks represent vehicles stuck in traffic on a 6x6 field. In this game one tries to have a special vehicle, the red car, escape the field by a gap in its border. However, vehicles are restricted in their movement as they can only move forwards or backwards, and cannot cross each other. In this work we search for features that are characteristic for hardness of a configuration within 6x6 Rush Hour. To perform this search the entire configuration space of 6x6 Rush Hour is computed and stored, and every configuration is solved. Surprisingly the results of this computation deviate from earlier obtained results[1] as more configurations are found. A new hardest configuration is found using a different definition of a move, and characteristics of the configuration space are analysed.

Title: On the Hardness of 6x6 Rush Hour
Author: Jelle van Assema, 10200363
Supervisor: Daan van den Berg
Email: Jelle.vanAssema@student.uva.nl, D.vandenBerg@uva.nl
Curriculum: Bachelor Kunstmatige Intelligentie
Number of Credits: 18 ECTS
Date: June 26, 2014
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam
Science Park 904, 1098 XH Amsterdam
http://www.science.uva.nl/home.cfm

Contents

1 Introduction

Rush Hour is a solitaire sliding block puzzle with simple rules. 6x6 Rush Hour is the commercial version and is played on a six by six field with two types of blocks, which from now on will be called *vehicles*, a *car* of length two and a *truck* of length three. Vehicles are placed on the field and can move forwards or backwards but not through other vehicles or over the edge of the field. On the field there exists one special vehicle: the *red car*. This red car is always placed on the third row from the top, and at the end of the third row there is a gap in the field's border which allows the red car to escape. Only the red car is allowed to escape the field, and once it has escaped the game is won. Thus the game is quite simple, one starts with a collection of vehicles on the field, which from now on will be called a *configuration*, then by performing a series of moves, where shorter series are deemed better, one tries to let the red car escape the field. The configuration where the red car is parked before the gap in the field is called a *solution* for the red car can then always escape the field within one move.

Simple as it may appear, solving a configuration, thus finding the shortest path to the nearest solution, proves to be quite difficult. In fact, deciding whether a configuration is solvable or not was proven to be a PSPACE-Complete problem[3]. Not only is the game difficult to solve, when presented with a configuration it is equally difficult to estimate how difficult it will be to solve. Difficulty of a configuration in Rush Hour is based on the length of the shortest path to the nearest solution, essentially the smallest number of moves needed to have the red car escape the field. Often configurations appear easy as they contain few vehicles or the red car is close to the exit, but prove difficult to solve. The opposite is also true, what is seemingly hard may prove to be remarkably easy. Studying the properties that influence the game's hardness, and how one can identify an easy or hard to solve configuration is the focus of this work. In order to study these properties the entire configuration space within 6x6 Rush Hour is computed and stored.

The contribution of this work is twofold. Firstly, attempted replication of the results obtained by Collette et al.[1] by generating all possible configurations for 6x6 Rush Hour. Secondly, knowledge on 6x6 Rush Hour such as the distribution of hardness among configurations, which could support the development of new solving or hard configuration generation algorithms for 6x6 Rush Hour.

2 Literature Review

Rush Hour is a game that has not received much attention from the computer science community unlike other games such as chess. This section provides an overview of the limited work on the puzzle.

Firstly, Flake et al.[3] examined a generalised version of Rush Hour; a version which is not limited to the standard field size and exit location. Flake et al. managed to prove that deciding whether a Rush Hour instance is solvable or not is NP-hard. After which they showed the general problem's PSPACE-completeness. Flake et al. mention that Rush Hour is the simplest game with the PSPACE-completeness property that they know of. Hearn et al.[5] went one step further, and managed to prove PSPACE-completeness of sliding block puzzles in the more general case.

The proof for NP-hard gives an indication that there does not exist any polynomial time algorithm that is able to find a solution for every Rush Hour configuration. Despite this indication Fernau et al.[2] considered the parametrised complexity of the generalised version of Rush Hour. They managed to show that as long as either the total number of moves or the total number of vehicles is bounded by a constant, solutions can be found in polynomial time. Hauptman et al.[4] created some heuristic functions through the use of genetic programming that are seemingly successful in comparison to human designed heuristic functions. Though these heuristic functions perform well on average, the performance of these functions on specific configurations are inconsistent. This inconsistency serves as an indication that some configurations are more difficult to solve than others or require different search strategies.

Finally, Collette et al.[1] focused on finding hard initial configurations for 6x6 Rush Hour. The shortest path to the nearest solution was calculated for all possible 6x6 configurations through the use of propositional logic and symbolic model-checking techniques within reasonable time. As a result, the study provided knowledge on the most difficult configurations within 6x6 Rush Hour and knowledge on the scope of the game. Collette et al. claim there exist 36 billion configurations in total of which 29.8 billion configurations are solvable. In this context, solvable means that for a configuration there exists a series of moves that leads to a configuration where the red car has left the field. After finding all solvable configurations, the configurations were kept in memory and could then be studied.

3 Research Question

Previous studies have shown the complexity of solving a configuration of Rush Hour and the difficulty of designing heuristic functions that help solve configurations. Especially the inconsistency in performance of these heuristic functions among different configurations is interesting. This is interesting as some configurations are apparently harder to solve than others, or some configurations benefit more from different solving strategies. A classification of hardness of a configuration could serve as help in the choosing of the applicable solving strategy. From this the question rises: For the game of Rush Hour, what features, if any, exist that are distinctive for hardness of a configuration? In order to provide an answer, hardness of a configuration needs to be defined. Earlier studies such as Collette et al.[1] and Hauptman et al.[4] used the length of the shortest path to the nearest solution for any given configuration as the measure of hardness for that configuration. This definition is what will be used as definition of hardness. Accepting this definition allows for the classification of hardness for each configuration and creates a gradation from easy to hard to solve configurations.

In order to determine the length of a path, the definition of a move has to be made. Collette et al. defines a move as moving one vehicle one cell on the field. This definition can be used for the measurement of hardness, yet it has a somewhat strange effect. As through this definition a configuration that requires moving four separate vehicles one cell is equally difficult as moving one vehicle four cells in one turn. Yet the planning required to move four vehicles is harder as vehicles can block each other, yet moving one vehicle four cells is simply doable if there are four cells in front or back of the vehicle that are not occupied. On top of this, the definition also finds a configuration with a nearly empty field and just the red car at maximum distance from the gap on the field more difficult than the same nearly empty field only now with the red car at minimum distance from the gap. Intuitively these two configurations are equally difficult when solving the configurations by hand, as one immediately sees that the red car can slide out of the gap. Often when playing this game in person, one does not slide a vehicle one cell at a time but multiple cells per slide. As such it seems logical to define a move as moving a vehicle for any number of cells on the field. This change in definition in comparison to earlier work does impact the degree for hardness for certain configurations as will be discussed later on.

4 Research Method

In order to search for features that are distinctive for hardness of a configuration, a sufficiently large set of configurations is needed that ranges across the scale of hardness. Collette et al.[1] managed to produce such a set, as this set contained every configuration that is solvable within the game. Although they did not store their results, they were able to produce and solve every configuration within reasonable time. The first step in this study is thus replicating the results of Collette.

4.1 Generating and Solving All Configurations

The solution Collette designed to tackle the problem of generating and solving all configurations relies on the propositional modelling of the graph of configurations where each node represents a configuration and a connection the direct transition from one configuration to the other. Collette then made use of symbolic model-checking techniques to explore this graph. Instead of directly copying the method Collette used, a new method inspired by Collette was designed. The main goal of this method is to construct a database which can be used to gain access to configurations on the basis of their hardness.

Starting with the previous results, Collette's results indicate that there are 10 billion solutions, 29.8 billion solvable configurations and 36 billion configurations in total. This should give an indication of the scope of the problem. Yet as these numbers appear large, Collette managed to generate and solve all 29.8 billion solvable configurations within ten hours on would now be considered a technologically outdated computer. Thus the time needed to solve and generate these configurations poses no threat to my study, however the space required to store these configurations does pose a problem. Just enumerating every configuration requires 35 bits of storage for every configuration, which would lead to a database of roughly 130GB in size. This size is not infeasible, yet fairly difficult to work with as this does not fit into memory of a regular computer and would thus result in very limited operating speed. Therefore, the decision has been made to investigate alternative methods of storing the database.

Storing all solvable configurations leads to a large and thus difficult to work with database. Yet the need to store all solvable configurations is not entirely clear, as when one configuration has a minimum distance to the nearest solution of 30, there are 29 configurations between it and the solution. Hence there are essentially 29 configurations that are easier and that can be recomputed from just that one configuration by solving the configuration. Essentially all configurations that are reachable from a configuration can be generated from just that one configuration. This can be done by performing all possible legal moves on that configuration

and by then exhaustively performing all legal moves on all the configurations that spawn after the application of these moves until no new configurations are found. This network of reachable configurations is henceforth denoted as a *cluster*. Based on this definition a configuration always belongs to one and only one cluster. A solvable cluster is a cluster that contains a solution and thus every configuration within that cluster is solvable.

Only storing solvable clusters leads to a significant decrease in space required, yet the generation of clusters is somewhat impractical. As each cluster needs to be found preferably just once and this is a difficult task to achieve. Simply generating all 29.8 billion solvable configurations and from those all clusters is infeasible as there are far fewer clusters than there are configurations. Thus there exists a need to specifically generate clusters. To generate clusters a deeper understanding of what a cluster entails is needed. Clusters are defined as a set of reachable configurations, as such all configurations within a cluster share quite some similarities such as the same number of cars and trucks on the field. Actually the only difference between the configurations in a cluster is the positions of the vehicles. Even the possible positions for a vehicle are limited, as vehicles cannot cross each other and can only move forwards or backwards. Thus when a car and a truck are placed in length on a column of a field, they can never swap places nor leave that column. The same is true for when a car and truck are placed on a row. Taking these characteristics a configuration can be represented as a collection of six rows and six columns where each row or column contain only the vehicles that are placed in length on the row or column. Rows and columns can even be generalised as they are essentially the same just interpreted differently. Accordingly, a *line* may be defined as a row or column in a configuration containing only the vehicles placed in length on the line. Thus a configuration is represented as a collection of twelve lines, where six lines are interpreted as rows and the remaining six as columns. Now there are only 24 lines in 6x6 Rush Hour and not all these lines can reach each other, for instance a line containing only a truck can never reach a line containing a car. A line containing a truck can reach every other line containing only a truck. Thus, sets of lines exist that are reachable from each other and this reachability is based solely on the way these lines are filled and in what order. The order is important as a line containing a car and a truck can never reach a line containing the same vehicles but in reversed order. A set of reachable lines will henceforth be called a *filling*. As a configuration is represented as twelve lines and each line belongs to a filling, a configuration can potentially only reach all configurations which share those fillings. However, this is not a suitable definition for a cluster as some configurations may share all fillings but are not reachable from each other. An example of how these configurations are not reachable is illustrated in figure 1. A set of twelve fillings is called a *cluster group*. A cluster group is a group

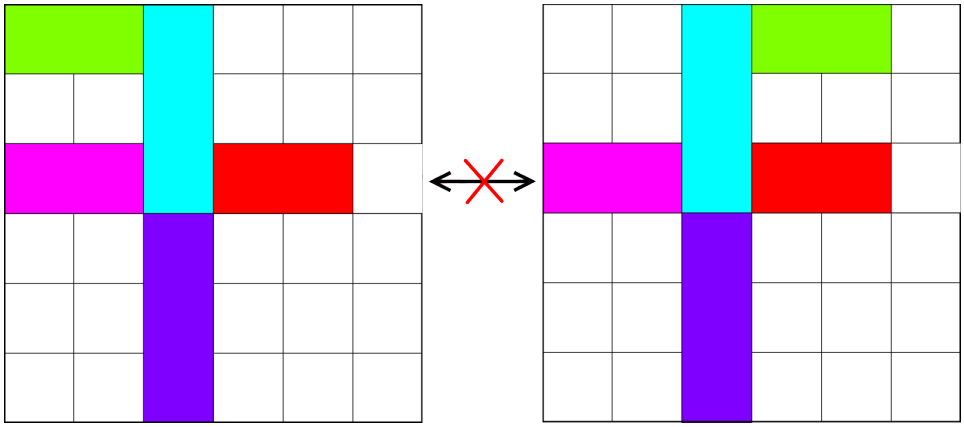

Figure 1: Two configurations within the same cluster group but not within the same cluster.

of clusters that all share the same twelve fillings and a cluster group can thus be represented as a collection of twelve lines. A solvable cluster group is a cluster group that contains a solvable cluster. 6x6 Rush Hour contains only eight fillings and therefore there cannot exist more than 8^{12} cluster groups. In fact by the rules of the game there can only exist $4*8^{11}$ solvable cluster groups as there must always be a red car placed on the third row from above. More importantly, the countability of these cluster groups has two main advantages for the generation of cluster groups. Firstly counting ensures finding all cluster groups. Secondly, when counting no cluster groups are found more than once.

4.1.1 Generating Solvable Cluster Groups

As cluster groups are countable, simply counting leads to finding every cluster group, however this includes cluster groups that cannot be made. For instance if a cluster group has only fillings with three cars, that would lead to a 6x6 field containing $12 * 3 = 36$ cars. This is impossible to make as cars would always overlap, because the cars take up more space than there is room on the field. Hence each cluster group has to be checked whether it can legally form a configuration. Here legally means, a configuration where vehicles do not overlap, and are all placed on the field. For the generation of solvable cluster groups it is beneficial to check if a cluster group can legally form a solution, as a solvable cluster group must contain at least one solution.

The generation of all cluster groups is essentially done by traversing a tree of depth twelve, where each node represents a filling and contains exactly eight

(which is the number of possible fillings within 6x6 Rush Hour) children and the leaves represent the cluster groups. Traversing all nodes is unnecessary as for some combinations of nodes no solution can be formed. For example seven fillings each containing three cars can never form a solution as the space occupied by the cars would exceed the space on the field. Hence further traversing the tree after having traversed these seven nodes is unnecessary. Simply checking the space occupied by the vehicles on the fillings over the space available has only limited success. For example a cluster group containing a filling for a column consisting of three cars and a filling for a row consisting of three cars can never form a solution, even though the space occupied by the vehicles is less than the space available. As rows do not hinder the movement nor placement of vehicles on other rows, and there exists an empty filling, picking six fillings which represent rows will always lead to at least one cluster group as picking the empty filling for each column creates a cluster group that can form a configuration. Essentially the same holds for columns, yet the third row contains the red car, thus for generating solvable cluster groups it is beneficial to start with rows. Because of this feature, pruning is impossible before depth six, as every node leads to at least one solvable cluster group. As a result, a lower bound is established: $4 * 8^5$ solvable cluster groups.

Pruning however is possible from depth seven as seen earlier. Preferably no nodes are traversed that lead to a cluster group that cannot form a solution. This can be achieved by first picking six fillings that represent rows. For each row all lines are retrieved. Upon picking a filling for a column at depth seven or higher the lines for the rows are filtered per column picked. In this context filtering means

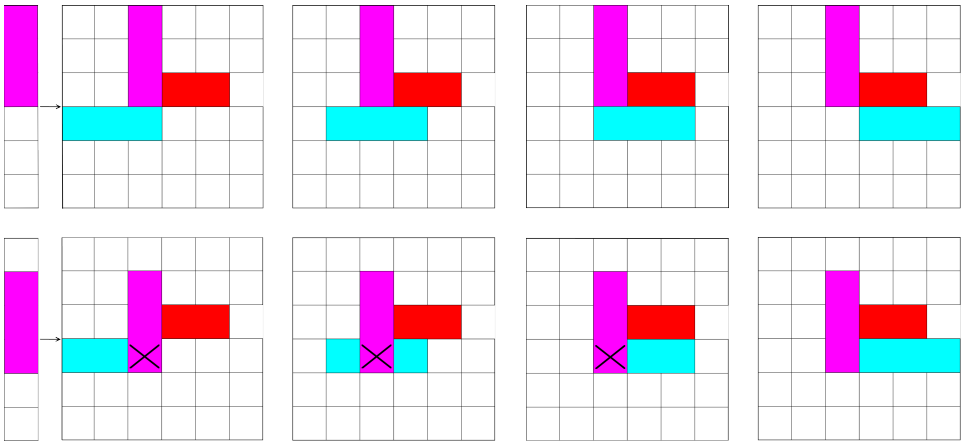

Figure 2: Illustration of how different lines within a filling can impact the number of configurations that can be made.

removing the lines that would overlap with the columnn. If for a row no lines are left, the branch is applicable for pruning. Yet there exists a problem, a column contains multiple lines and not every line filters the same lines from the rows. This is illustrated in figure 2. The problem can be solved by having every connection in the tree represent a specific line picked from the node from which it spawned. Essentially this creates a tree where nodes can be reached in multiple ways depending on how many lines a filling contains. Having solved this problem, every leave found must thus contain a solvable cluster group for there exists a set of lines that can form a solution. In contrast there exists numerous paths to the same nodes, and therefore nodes are traversed multiple times and thus leaves are found numerous times. This problem is easily solved by simply ignoring all leaves that have been traversed and ignoring all nodes that have traversed their entire subtree, and thus have found all leaves.

4.1.2 Generating Solvable Configurations

Now remains the task of finding all solvable configurations. The ability to find all solvable cluster groups helps greatly with this task. For all solvable configurations must reside within a solvable cluster group and can thus be found by finding all configurations within a cluster group. Similarly to finding all solvable cluster groups this task can be achieved by simply looking at all possible combinations of lines that can be made within a cluster group. However, as figure 1 illustrates, this can also find configurations that are unsolvable, because not all configurations within a cluster group are reachable from each other. Furthermore, this technique does not find the shortest path to the nearest solution for each configuration, and would thus require further computation to classify each configuration on hardness. Hence an algorithm which can explore the entire configuration space within a cluster group and which is guaranteed to find the best solution for each configuration is preferred. Both of these properties are found in a simple breadth-first search algorithm. Given a configuration, this configuration can be solved by simply finding all directly reachable configurations and then exhaustively searching for directly reachable configurations for each configuration found in order of which they where found until a solution is found. Sadly this solves just one configuration while traversing many. However reversed breadth-first search remedies this problem. For reversed breadth-first search one starts with all solutions for a cluster group, then for each solution search for directly reachable configurations, and finally by exhaustively searching for directly reachable configurations for each configuration found in the order in which they are found, one finds all solvable configurations and the shortest path to the nearest solution for each configuration.

Firstly, finding all solutions for a cluster group. This can be achieved by taking

all lines for each filling of the cluster group and only the lines with a car parked at the end for the the filling that represents the third row from above, for that is the row with the red car. By creating all possible combinations that also form a correct configuration all solutions for the cluster group are found. Similarly to finding all cluster groups, this can be done be traversing a tree, where each node represents a line and each leave a configuration. If one starts with picking lines for rows, then pruning is possible in the tree from depth seven as the placement of rows does not hinder the placement of other rows. Pruning is possible for when a column does not fit with the current selection of rows, no configuration can be found by picking subsequent columns.

Secondly, finding all directly reachable configurations from a configuration. This can be achieved by changing one line from a configuration and replacing it with another from the same filling where the placement of one and only one vehicle differs from the original line. However this method also produces unreachable configurations as illustrated in figure 1. Thus when replacing a line for another there has to be checked whether that line is reachable given the configuration. This reachability check can be done by checking the rows or columns that reside in between the original location of the moved vehicle and that of the end location on vehicles. For if such a column or row contains a vehicle the line is not reachable from the original line.

Finally, given all solutions and the ability to find all directly reachable configurations from a configuration, the implementation of reversed breadth-first search is just an implementation of a standard breadth-first search. Only in reversed breadth-first search one starts with all solutions and one keeps searching till all configurations are found. During this reversed breadth-first search one essentially explores a tree where each node represents a configuration and the root nodes represent the solutions. In this tree the depth of a node indicates the length of the shortest path to the nearest solution, and thus hardness is defined for every configuration. For the generation of this tree all that needs to be taken into account is that a configuration can always directly return to its parent node or nodes, and sometimes indirectly to its parent or ancestor nodes. Therefore when building this tree through breadth-first search, already visited configurations need to be filtered out and not visited again.

4.1.3 Building The Database

As explained earlier, storing all configurations is infeasible as the size of the database would hinder its performance to great extend. Therefore a database is constructed containing only cluster groups. Using the solvable configuration generation algorithm described in the previous section all solvable configurations can be found from these cluster groups. Preliminary results showed that finding and solving all

configurations within a cluster group could be done in the scale of tens of milliseconds depending on the size of the cluster group. For Collette et al.[1] claimed there are $29.8 * 10^9$ solvable configurations, an early estimation of the time required to compute and solve all solvable configurations was around one week. Given this estimation and the time at hand, it was deemed necessary to store some extra features other than just the fillings which it contains. Most important among these features is the maximum distance, that is the longest shortest path from a configuration within the cluster group to a solution within a cluster group. This feature essentially allows quick identification of cluster groups that contain hard to solve or only easy to solve configurations. Besides this feature, the overall size, that is the number of solvable configurations within the cluster group, and the number of solutions within the cluster group were stored as well.

5 Results: All Solvable Configurations

Using the solvable configuration generation and solvable cluster group generation algorithms, all solvable configurations were found, solved and subsequently stored within roughly 100 hours. This performance was obtained on an Intel(R) Core i5-2500K(TM) CPU 3.3GHz while using less than 200MB of ram for the generation algorithms, and around 1.5GB of ram for the database. The algorithms were implemented in Java, and the database was created in MySQL with InnoDB as storage engine. The size of the database containing just the cluster groups was roughly 3GB.

5.1 Different Results

The results from the study by Collette et al. were not obtained. The database contains $31,501,642,578$ solvable configurations of which $10,275,383,941$ are solutions. Yet Collette et al. found only $29.8 * 10^9$ solvable configurations of which around $10 * 10^9$ are solutions. As such the results of this work deviate roughly 5.7% and 2.8% respectively. An explanation for this difference is not easily found. Mainly for Collette et al. have not reported how they managed to obtain all possible solutions, which is crucial as their method, similarly to the method used in this work, rests on breadth-first search from the solutions.

When the difference in results was found, an algorithm was designed to try and replicate the other results Collette et al. mention: $36 * 10^9$ legal configurations of which $7 * 10^9$ are unsolvable. This algorithm is very similar to the algorithm described to generate solutions, only now the search space is not limited to a cluster group but to the entire configuration space within 6x6 Rush Hour. As such, all configurations cannot be kept in memory. This however does not pose a problem as they do not need to be. One simply applies the solution generation algorithm on every possible filling while not limiting the line on which the red car resides to only lines with a car at the end. Instead of collecting all configurations found, each configuration is counted. This is possible since the solution generation algorithm does not find a configuration more than once.

The problem here is the definition of legal. Collette et al. never explicitly mention whether they deem it legal for any other vehicle than the red car to leave the field. Assuming they deem it illegal, then there exist $40,148,868,698$ legal configurations, which is roughly 11.5% more. If they deem it legal there would only exist more configurations. With $31,501,642,578$ solvable configurations that would mean there exist $8,647,226,120$ unsolvable configurations, which is roughly 23.5% more than reported. Which means that roughly 21.5% of the entire configuration space is unsolvable.

With the differences found between every result, a possible explanation was sought. A most likely cause is missed solutions, and therefore missed clusters. As Collette et al. did not store their results checking for missed clusters is not feasible. What is possible however is finding the clusters within each cluster group for every cluster group, which provides insight in the scope of how missed solutions lead to missed clusters. This can essentially be done by applying the solution generation algorithm, still limited to the scope of a cluster group, only now no longer limited to lines for the red car where there is a car at the end. Using this algorithm all legal configurations within a cluster group are found. For every legal configuration found a cluster is made by applying breadth-first search on the configuration until no new configurations are found, unless the legal configuration is already within a cluster then it is simply ignored. As such all clusters are found, yet not every configuration within every cluster is solved. Firstly, some configurations within clusters of a solvable cluster group cannot be solved as they lie within an unsolvable cluster. So solving every configuration within every cluster group is impossible, however the unsolvable clusters within these cluster groups are easily identified by the lack of solutions within them. Therefore all that needs to happen is to identify the solvable clusters, and generate them again by starting with the solutions that were found during the first generation, and simply applying the cluster group generation algorithm on these solutions.

Applying this algorithm and storing the results in the same database lead to an increase in size of roughly 12GB. By doing so, access was gained to not only all solvable cluster groups, but also all solvable and unsolvable clusters within them. Other than simply having access to more data on 6x6 Rush Hour this also gives insight in the scope of how missing solutions can impact the number of configurations found in total. On average 85% of the solutions within a solvable cluster groups lie within the cluster with the most configurations. Moreover 85% of the configurations lie within the cluster with the most configurations of a solvable cluster group. Thus when one picks one solution for every cluster group and keeps just the cluster formed by that solution atleast roughly 70% of the total number of configurations are found. The results by Collette et al. lie within this scope, thus it is definitely not proven that such an error was made but it is seemingly plausible.

5.2 New Hardest Configuration

In accordance with the definition for a move used in this work, the claimed hardest configuration by Collette et al.[1] is not deemed the hardest. In fact, the hardest configuration by Collette using the definition for a move used in this work shares a spot with 21 other configuration in four clusters as third hardest at a distance of 49 moves from the nearest solution. The hardest configuration sits at a distance

of 51 moves from the nearest solution and is illustrated in figure 3. Although this configuration has never been reported as the hardest configuration, it had already been found by Collette et al. and was reported at a distance of 83 moves from the nearest solution, where moves are measured using their definition of a move. Interestingly, the number of configurations reachable from the hardest configuration is 4780. This is relatively small in comparison to the number of configurations reachable from the hardest configuration claimed by Collette et al. which is 24132.

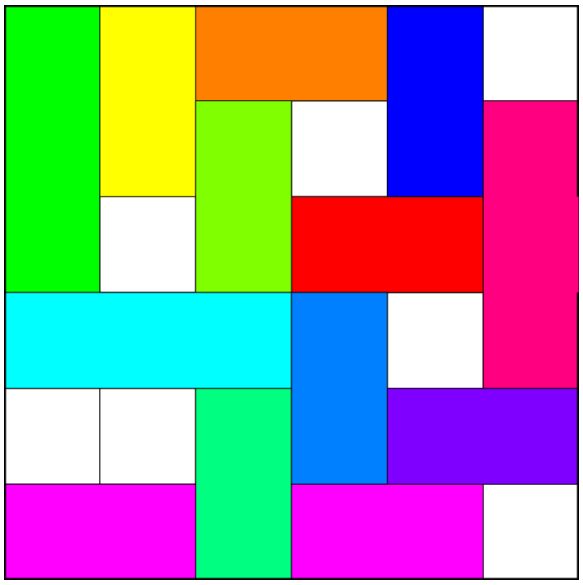

Figure 3: One of three configurations with the longest minimal path of 51 moves to the nearest solution within 6x6 Rush Hour. The remaining two configurations reside within the same cluster.

5.3 Influence On Maximum Distance

When looking for features that have an impact on the maximum longest minimal path that a configuration can have from a solution, a logical candidate would be the size of the cluster group and cluster, that is the number of configurations within it. As a small size limits the maximum distance as for a path of length n there need to exist at least $n + 1$ configurations within the cluster group. Though a large cluster group does not mean the maximum distance, that is the maximum longest minimal path that a configuration has from a solution within that cluster group, is large. For instance there exists a cluster group with 541934 configurations, yet the maximum distance within that cluster group is 15. A large number of configurations within a

cluster group indicates that the cluster group contains fillings that have many lines, mostly fairly empty fillings containing only one car or truck. As such fillings do not have a large effect on the movement of vehicles on the field, generally these clusters are very large yet the longest minimal path from a configuration to the nearest solution is small. This effect is visible in figure 4. Clearly a small cluster group is not beneficial on average for a large maximum distance, though the gain in maximum distance comes to a hold at roughly 20000 configurations per cluster group. At that point other factors come into play, such as the number of solutions within the cluster group.

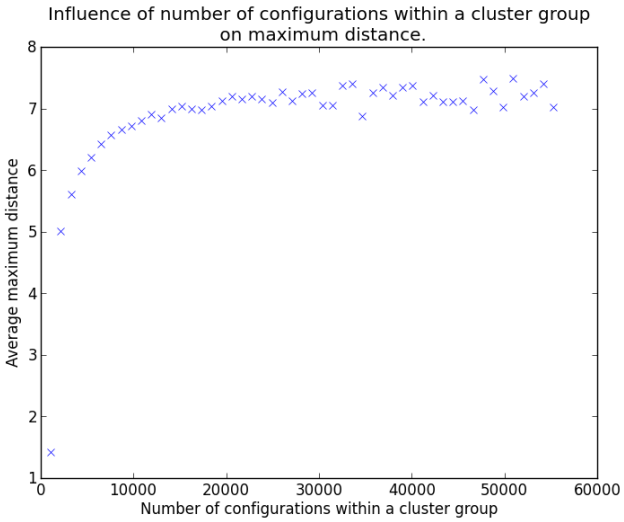

Figure 4: Influence of number of configurations within a cluster group on the maximum distance. Each point represents a bin of cluster groups. There is a bin for every 1000 configurations within a cluster group.

Similarly to the cluster group size, the number of vehicles within a cluster group strikes out as an interesting feature. As very few vehicles on the field would seriously hinder the maximum distance. Though many vehicles also indicate a small maximum distance. For instance three cars on every row and nothing on every column will always have a maximum distance of one. This effect is illustrated in figure 5. As can be seen in the figure, there exists a maximum at roughly eleven to thirteen vehicles in the cluster group. Moreover, all configurations with a distance of forty or more to the nearest solution have ten to fourteen vehicles on the field.

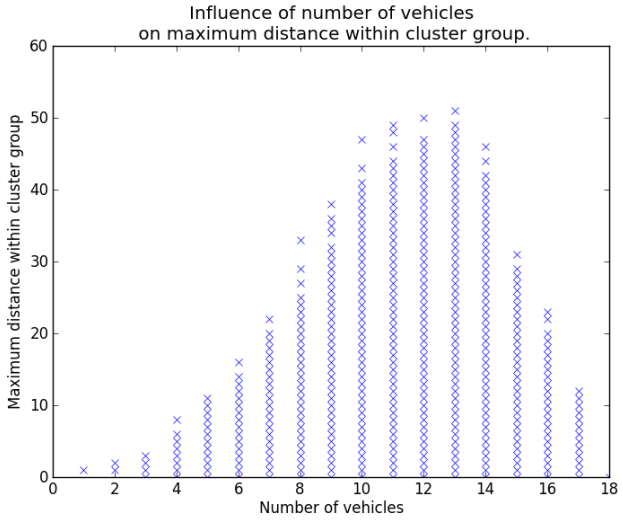

Figure 5: Influence of number of vehicles within a cluster group on the maximum distance. Each point represents a cluster group.

5.4 Identifying Unsolvable Configurations

As Flake et al.[3] showed, deciding whether a configuration is solvable or not is a PSPACE-complete problem. The 6x6 version of Rush Hour contains $40,148,868,698$ legal configurations of which $8,647,226,120$ configurations are unsolvable. $34,811,208,267$ configurations lie within $68,478,733$ solvable cluster groups which contains $172,234,727$ clusters. This means that $5,337,660,431$ unsolvable configurations do not lie within a solvable cluster group and as such lie within an unsolvable cluster group. Thus for roughly 61.7% of the unsolvable configurations no solutions can be found as the chosen fillings for those configurations simply cannot form a solution. Which in turn means that when one picks a configuration at random from the entire configuration space there is a 21.5% chance that that configuration is unsolvable. Yet only searching for a solution and not the path to the solution reduces that percentage to just 8.2% when a solution is found. Sadly the solution generation algorithm introduced in this work still requires exponential time. Though the search space for a solution given a cluster group is severely limited in comparison to the search space for finding the shortest path from a configuration to the solution within the same given cluster group. Because the line containing the red car must have a car at the end for a configuration to be a solution, which is a subset of the number of configurations within that cluster group.

5.5 Distributions

With $34,811,208,267$ solvable configurations and $172,234,727$ clusters, the average number of configurations per cluster that lies within a solvable cluster group is 202.1 with a variance of $2,370,214.12$. The minimum number of configurations within a cluster is one. That is a completely frozen configuration, for instance all empty lines except the line with the red car which has three cars. The maximum number of configurations within a cluster and cluster group is 541934. Thus there exists a set of 541934 reachable configurations. This cluster with 541934 configurations has the same filling for each line, that is one car. The distribution of the number of configurations per cluster is illustrated in figure 6. This shows the lack of clusters with many configurations and the large presence of very small clusters. Striking is also the lack of any cluster with 360000 to 420000 configurations while there exists a few clusters with more than 420000 configurations. Need be noted that the largest cluster, the one with 541934 configurations, is barely visible in the figure thanks to the logarithmic scale.

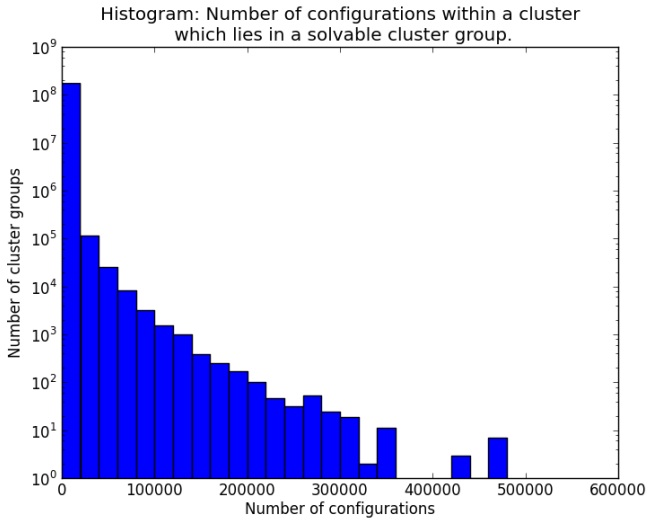

Figure 6: The distribution of configurations over clusters that lie within a solvable cluster group.

With $68,478,733$ solvable cluster groups which contain $172,234,727$ clusters, the average number of clusters within a cluster group is 2.52 with a variance of 1.62. The minimum number of clusters per solvable cluster group is obviously one. The maximum however is 136. There exist fourteen cluster groups with 136 clusters, and they are all very similar. For all these clusters have a line that is

completely filled, thus that line has either two trucks or three cars. This line is always the third of fourth row or column. If the completely filled line is a column than each row contains one car, three columns contain one car, and one column contains two cars. Conversely if the filled line is a row. The distribution of the number of clusters per cluster group is illustrated in figure 7. As can be seen, there exist many cluster groups with few clusters and very little cluster groups with many clusters. Also visible in the figure are gaps as there are no cluster groups with 100 to 105 clusters and none with 115 to 125 clusters. While there exists a cluster group with 136 clusters.

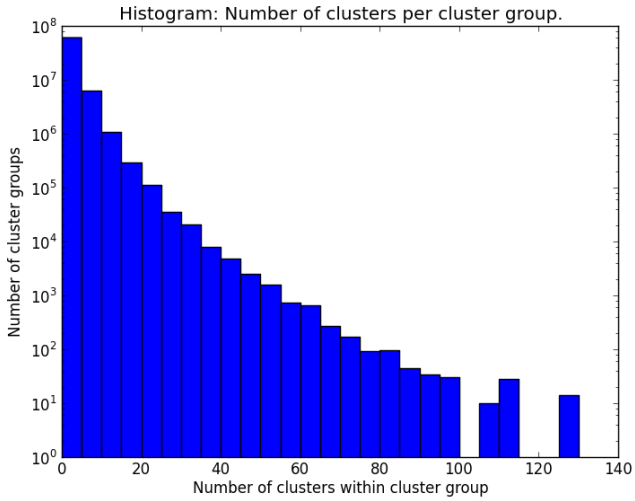

Figure 7: The distribution of clusters over solvable cluster groups.

As mentioned earlier the maximum distance from a configuration to its nearest solution is 51 and there only exists one cluster with that maximum distance. In fact, as can be seen in figure 8, configurations with large maximum distances are very rare. This illustrates the difficulty of finding hard configurations, because less than two percent of the configurations have a distance of ten or more to its nearest solution. There is also a trend visible. The line in figure 8 is fitted through the logarithm of the number of cluster groups per maximum distance within that cluster group. This trend indicates that it is possible to predict the maximum distance of 6x6 Rush Hour and possibly larger versions of Rush Hour.

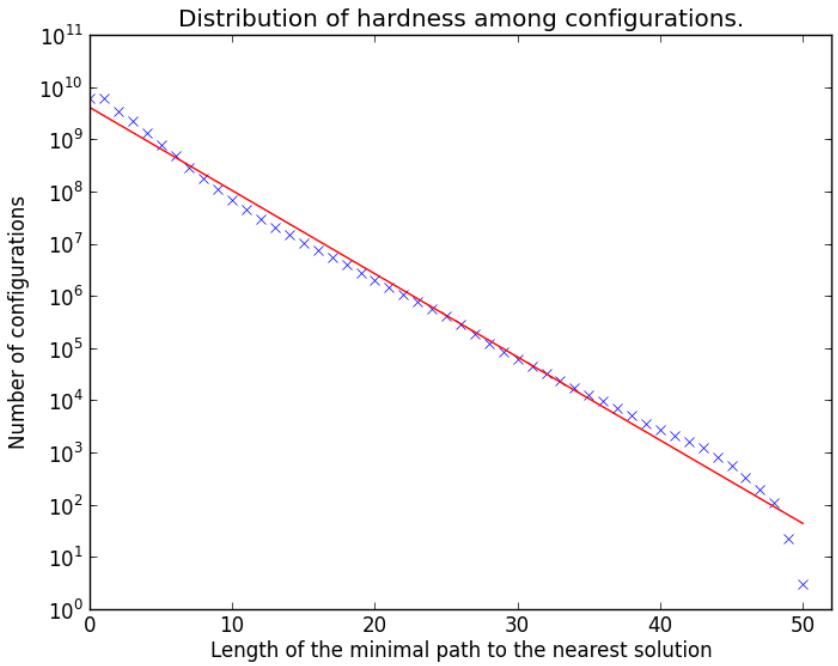

Figure 8: The distribution of hardness in the entire configuration space.

6 Conclusion

An attempt was made to replicate the results by Collette et al.[1]. The results however do not match as more configurations were found in total. An explanation for the difference remains absent. As all the results were stored, access was gained to the entire configuration space within 6x6 Rush Hour. With this access and another definition for a move, a new hardest configuration was found at a distance of 51 from its nearest solution. Moreover access to the entire configuration space shed light on some previously unknown statistics such as the distribution of hardness over the entire configuration space within 6x6 Rush Hour.

Furthermore in this work an attempt was made to answer the following question: For the game of Rush Hour, what features, if any, exist that are distinctive for hardness of a configuration? Though some success was achieved as clearly the number of vehicles and number of configurations in a cluster group impact the hardness, this question is not completely answered. After the database of cluster groups and clusters had been constructed the search began for features which are distinctive for the hardness of a configuration. The initial search for these features included many more features such as the distance from the red car to the gap or the number of solutions in a cluster group. Yet not many features showed any promise and were quickly discarded. During the search for these distinctive features it became apparent that in fact the definition of hardness was lacking and did not fully describe hardness. In this work just the distance of the shortest path to the nearest solution is used to classify a configuration on hardness. This however does not grasp what hardness is in essence, which is the difficulty of solving a configuration. Figure 9 and 10 show two configurations both with the same minimal distance to the nearest solution. Though one is substantially easier to solve as very few configurations are reachable from the configuration and very few configurations are directly reachable for each configuration that is reachable. When one explores the graph underlying the game this difference becomes more apparent. In the underlying graph each node represents a configuration and each connection the direct transition from one configuration to another, which is essentially performing a move. These graphs are illustrated for both configurations mentioned earlier in figure 11 and 12 respectively. These graphs give an indication that not just the minimum distance from a configuration to a solution defines the hardness but also the number of reachable configurations from that configuration, the branch factor of the configurations that are on the path to the nearest solution, and finally the solution density within the cluster. These four characteristics would lead to a more fitting description of hardness. As very few reachable configurations make for a game that is quite easily navigated through without any need for directed search. A small branch factor of the configurations that are on the path to the nearest solution

gives nearly no choices for the player or solving algorithm to make, essentially this creates a narrow tunnel within the graph where not many mistakes can be made. Finally, a high solution density generally indicates a very small maximum distance within the cluster. How this characterisation of hardness should be defined is open for future study.

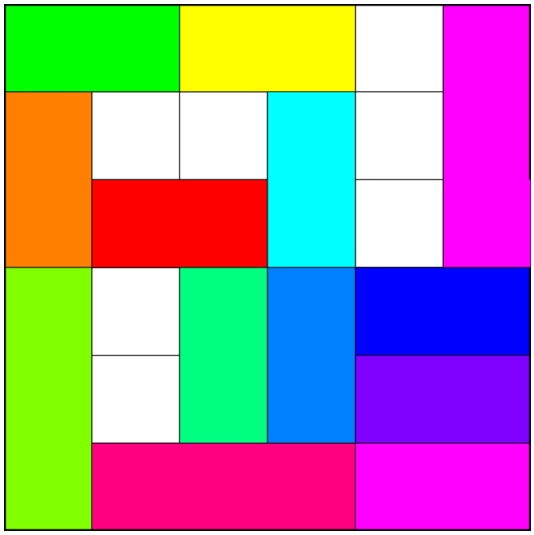

Figure 9: A configuration that has a pathlength of 31 to the nearest solution, but is fairly easy to solve.

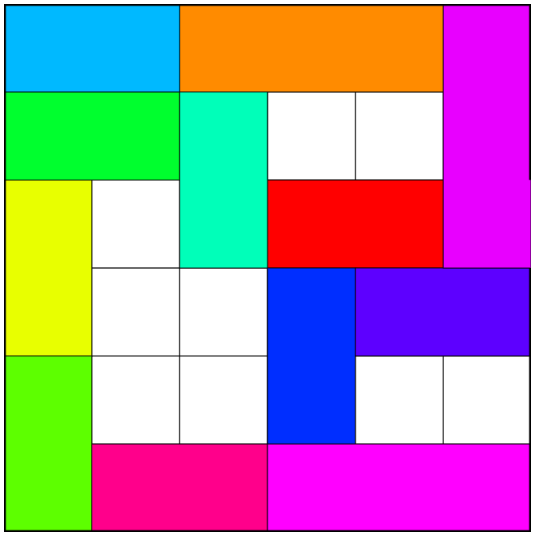

Figure 10: A configuration that has a pathlength of 31 to the nearest solution, but is relatively hard to solve.

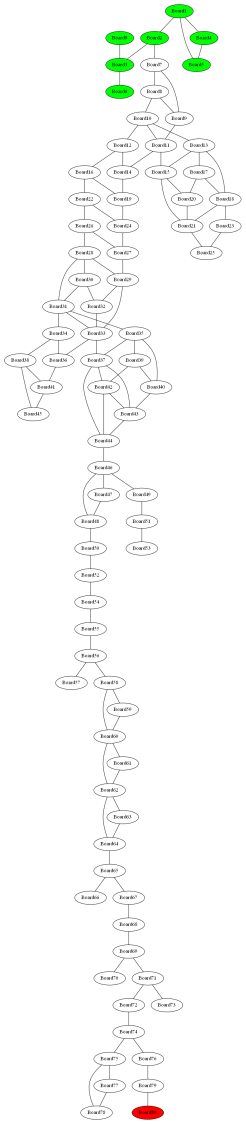

Figure 11: Graph of a cluster group of the configuration depicted in figure 9 where each node represents a configuration and each connection a move. Maximum distance within this cluster is 31. The green nodes indicate solutions and the red node the hardest configuration and the configuration depicted in figure 9.

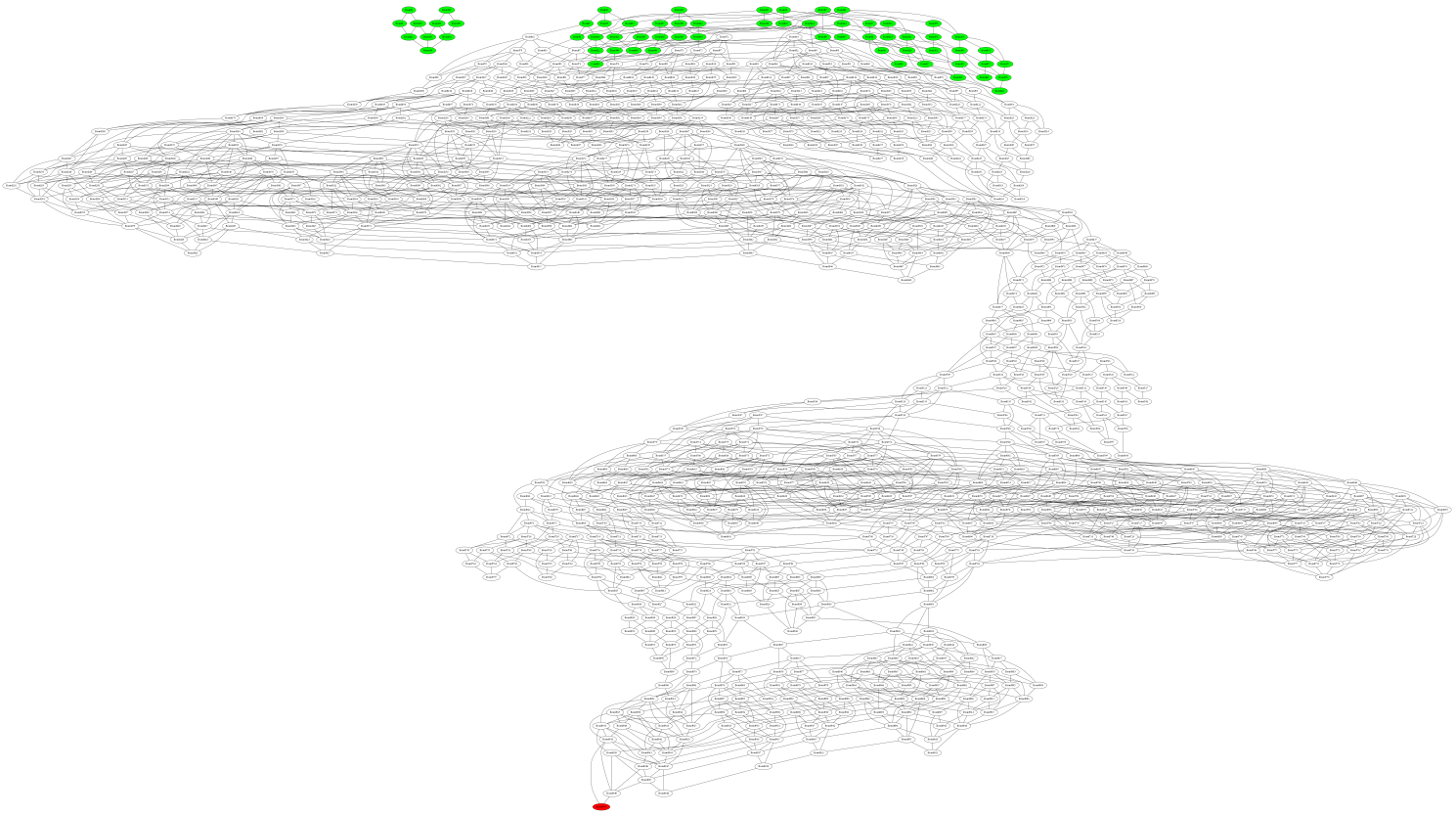

Figure 12: Graph of a cluster group of the configuration depicted in figure 10 where each node represents a configuration and each connection a move. Maximum distance within this cluster is 31. The green nodes indicate solutions and the red node the hardest configuration and the configuration depicted in figure 10.

References

[1] Sébastien Collette, Jean-François Raskin, and Frédéric Servais. On the Symbolic Computation of the Hardest Configurations of the RUSH HOUR Game. In H.Jaap Herik, Paolo Ciancarini, and H.H.L.M.(Jeroen) Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 220–233. Springer Berlin Heidelberg, 2007. 2, 4, 5, 6, 7, 13, 15, 22

[2] H. Fernau, T. Hagerup, N. Nishimura, P. Ragde, and K. Reinhardt. On the parameterized complexity of a generalized Rush Hour puzzle, 2003. 5

[3] Gary William Flake and Eric B. Baum. Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1–2):895 – 911, 2002. 4, 5, 18

[4] Ami Hauptman, Achiya Elyasaf, Moshe Sipper, and Assaf Karmon. GP-rush: Using Genetic Programming to Evolve Solvers for the Rush Hour Puzzle. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 955–962, New York, NY, USA, 2009. ACM. 5, 6

[5] Robert A. Hearn and Erik D. Demaine. PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation. *CoRR*, cs.CC/0205005, 2002. 5