A SURVEY OF NP-COMPLETE PUZZLES

Graham Kendall *Andrew Parkes* *Kristian Spoerer*[1]

Nottingham, UK

ABSTRACT

Single-player games (often called puzzles) have received considerable attention from the scientific community. Consequently, interesting insights into some puzzles, and into the approaches for solving them, have emerged. However, many puzzles have been neglected, possibly because they are unknown to many people. In this article, we survey NP-Complete puzzles in the hope of motivating further research in this fascinating area, particularly for those puzzles which have received little scientific attention to date.

1. INTRODUCTION

This article provides a survey of $puzzles$ that are contained in the set of those NP-Complete (Garey and Johnson, 1979) puzzles that have a single decision maker. We use the term *puzzle* to refer to single-player games which are enjoyable to play. By definition a puzzle should have a solution which (1) is aesthetically pleasing and (2) gives the user satisfaction in reaching it. We admit that this is rather a loose definition. A more formal definition is difficult as any other definition would only express our subjective view and would be open to debate. Therefore, we decided upon the above definition which captures our perception of a single player puzzle. We provide further comments in the conclusions. Puzzles have been the topic of artificial intelligence research for some time (Robertson and Munro, 1978; Hearn, 2006) and this survey is intended to motivate further research in this area by providing a starting point for the interested researcher.

We discuss 24 puzzles presented in alphabetical order. Two puzzles (Rubik's Cube and the Towers of Hanoi) are discussed separately. For each puzzle, a brief description of the rules and a reference to its complexity is provided. At the end of each puzzle description we summarize the decision problem that was used to prove its NP-Completeness. Note that some of the references are to URLs which suggests that there has been work directed towards these puzzles, but the work has not been presented for scientific peer review. It is our hope that this article will start to address the small number of published works, by encouraging authors to submit their research to journals such as this.

A brief description of the puzzle is included so that the reader understands it. There is often a reference to some software so that the puzzle can be attempted which might give an appreciation of the subtleties involved. Furthermore, the reader might gain an insight into the similarities between these hard puzzles. For example, the implicit hidden constraints that are only revealed once a puzzle is attempted.

A brief outline of some of the work that has been published about each puzzle is also provided. These works tend to apply search algorithms (see, for example, Russell and Norvig (2003)) to the puzzles. We have not attempted to include all of the research for each puzzle. Our motivation for briefly outlining this research is to provide references to the texts that can be used as starting points for future work on these puzzles.

Erik Demaine has completed some work in the area of Combinatorial Games[2]. We would like to recommend this work where it details the complexity of some two-player games and also some single-player puzzles. This work can be regarded as an extension of Demaine's (2001) survey of puzzles, and also an extension of Eppstein's survey of puzzles[3].

[1]School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK. Email: {gxk,ajp,kts}@cs.nott.ac.uk

[2]See http://theory.lcs.mit.edu/~edemaine/games/.

[3]See http://www.ics.uci.edu/~eppstein/cgt/hard.html.

2. COMPUTATIONAL COMPLEXITY

In this section, we provide an informal description of NP-Completeness. The treatment is informal by design as there are many other more formal treatments of this area, with Garey and Johnson (1979) being the classic text. For the interested reader we refer to the seminal work by Stephen Cook (1971) and also his short article (Cook, 1984) which provides an excellent discussion. In fact, the opening of the article succinctly shows the problem faced when considering computational complexity: "*In general, it is much harder to find a solution to a problem than to recognize one when it is presented.*"

We are concerned with decision problems. That is, those problems that can be answered by Yes or No. This might appear to be quite a restriction on the type of problems we can consider but, in fact, most problems can be reduced to a decision problem. For example, the intuitive way to consider the Travelling Salesman Problem (TSP) is to phrase it as "*Given a TSP instance, return the shortest (optimal) route*." In fact, we can turn this into a decision problem by asking "*Given a TSP instance, is there a tour of length L, or less?*" and then we can keep reducing *L* until we receive a negative answer. More efficiently, we can carry out a binary search on *L* until we find the optimal solution.

The P (Polynomial) class of problems contain the decision problems that can be solved on a deterministic sequential machine in time that is polynomial to the size of the input. In some sense, these are seen as *easy* problems.

The NP (Non-deterministic Polynomial) class of problems are those problems that can have a correct solution *verified* in polynomial time. For example, for the TSP, once we are presented with a tour, it is easy to verify (in polynomial time) that the tour is of a given length. However, although the TSP (and all the problems in NP) can have a solution *verified* in polynomial time, nobody knows of any algorithm that can *solve* the problem in polynomial time. If we were able to show that P = NP (i.e., all NP problems can be solved in polynomial time), then we know that we can solve a wide range of *difficult* problems in polynomial time. NP problems are known as Non-deterministic Polynomial because if we had a non-deterministic machine, which was able to *guess* correctly at each decision point, it could solve the problem in polynomial time, which is the same as verifying the solution. Showing that P = NP (or not) is one of the most important open questions in theoretical computer science and, in fact, is one of the seven most important problems across all of mathematics according to Devlin (2005). However, it is widely believed that P $\neq$ NP.

NP-Complete problems are considered to be the hardest problems in NP. The definition of an NP-Complete problem is that it is in NP and every other problem in NP is reducible to it. By reducible we mean that there is a polynomial time algorithm that can transform an instance of one problem to another. These properties mean that if we ever find an efficient (polynomial time) algorithm for *any* NP-Complete problem, then we have an efficient algorithm for solving all NP-Complete problems, as we have a way of transforming one problem instance to another, in polynomial time. To show that a problem is NP-Complete it is sufficient to show that it is in NP and can be transformed from another NP-Complete problem by a polynomial bounded algorithm.

Each of the 24 puzzles below have all been shown to be NP-Complete (with the exception of Rush Hour, which is PSPACE-Complete). However, we should emphasize that any problem that has a finite problem space cannot be NP-Complete as we could solve the problem in constant time. For example, if we limit Cryptarithms (see below) to just the decimal digits then these instances are not themselves NP-Complete. Similarly, the 15-puzzle is not NP-Complete but the generalization to the n-Puzzle is.

3. THE PUZZLES

Below we describe 24 puzzles. They are given in alphabetical order.

(1) Blocks World is played on a table, which we can think of as being infinite in size. There are a finite number of blocks. Every block sits on top of exactly one other block or sits on the table. Any block that does not have another block on top of it is known as *clear*. An action moves a clear block from its current position, and puts it either directly onto the table or on top of a clear block. The difficulty of the task lies in the fact that you can neither move a block that has another block on top of it, nor can you place a block on top of a block which is already covered by another block.

An instance is a description of the blocks in some arbitrary arrangement, and a description of the blocks in their goal positions. A solution is an ordering of moves from the initial arrangement to the goal. The optimal solution is the one with fewest moves. As an example, you may be presented with a set of square, rectangular, and triangular blocks. The blocks could be arranged in some arbitrary way, including some being stacked on others. The goal state could be an arrangement such that the blocks resemble a house.

Finding a solution to Blocks World that is no more moves than some bound, has been shown to be NP-Complete by Gupta and Nau (1992). Blocks World has recently been studied by Slaney and Thiebaux (2001). They advocate the use of toy problems such as Blocks World on the condition that they are properly understood. Therefore, they offer knowledge about Blocks World at a sufficient level that it can be used as a benchmark and so that it can be successfully handled. In particular, they describe how to generate random instances that can be used for experiments. They also detail a number of algorithms that produce solutions for Blocks World along with their time complexity, and they explore the structure of hard and easy instances. In Blocks World the table is not limited in size, which means that for any starting arrangement of blocks, all of those that are out of place can be moved to the table and then moved one-by-one to their goal position. Out of place blocks must be moved at least once from their initial position, in order to place them correctly, even in the optimal solution. Thus, a simple algorithm can find some arbitrary near-optimal solution that is at most twice the optimal length in time that is linear over the number of blocks (Slaney and Thiebaux, 2001). Blocks World might be restricted to a more difficult problem by placing a limit on the size of the table so that it cannot hold all of the blocks, at which point finding *any* solution might be hard. In regular Blocks World we know that the shortest possible solution must move all out of place blocks to their correct positions. Therefore, finding the optimal solution involves finding the minimum number of extra moves that do not put blocks in their correct positions. Further literature on Blocks World can be found in Slaney and Thiebaux (2001). The interested reader can find more information at http://users.rsise.anu.edu.au/~jks/bw.html. This site provides functionality to generate problem instances and their solutions and also provides access to some of the literature.

DECISION QUESTION (Gupta and Nau, 1992): Given an Elementary Blocks World (EBW), problem (*I*,*G*), and an integer $L > 0$, is there a plan for this problem of length *L* or less?

Here EBW is an ordered pair B = (*I*,*G*), *I* is the initial state, and *G* is a goal state. *B* is solvable if there is a sequence of moves that moves the blocks from the initial state to the goal state.

(2) Clickomania (a variation is known as SameGame) is played on a grid of c columns and r rows. The grid is initially filled with differently coloured square stones, comprising k colours. Groups are formed by stones of the same colour of which the edges are touching. A move deletes a group that contains at least two stones. Stones are continuously pulled downwards until they touch either the bottom of the grid or another stone, so that any gaps made by deleting a group are filled by any stones above it. When a column is deleted all of the stones to the left and right of it move together to fill the space. An instance is a description of the grid that contains a stone in every position. A solution is successful if it removes every single stone. Biedl *et al.* (2002) show the following. Deciding the solvability of 1-column (or 1-row) 2-colour Clickomania can be done in linear time. Deciding the solvability of 2-column, 5-colour Clickomania is NP-Complete. Deciding the solvability of 5-column 3-colour Clickomania is also NP-Complete. Clickomania has not received any other scientific interest, at least as far as the authors are aware. The interested reader can visit http://www.clickomania.ch/click/, which provides a downloadable version of the game, and http://theory.lcs.mit.edu/~edemaine/clickomania/, which also describes some of the game's history. Figure 1 shows an example of a Clickomania puzzle[4].

DECISION QUESTION (Biedl *et al.*, 2002): Is a given instance of the problem solvable: can all the blocks be removed? (the optimisation version is stated as follows: find the maximum number of blocks that can be removed from a given instance?).

We also refer the reader to Schadd *et al.* (2008) who prove that SameGame is NP-Complete by a reduction from Clickomania with 5-colours and 2-columns.

(3) Corral Puzzle is played on an $m \times n$ grid. Some of the squares contain a number. The goal is to find a closed

[4]From Galaxy of Mahjongg 2, see http://www.greenstreetgames.com/

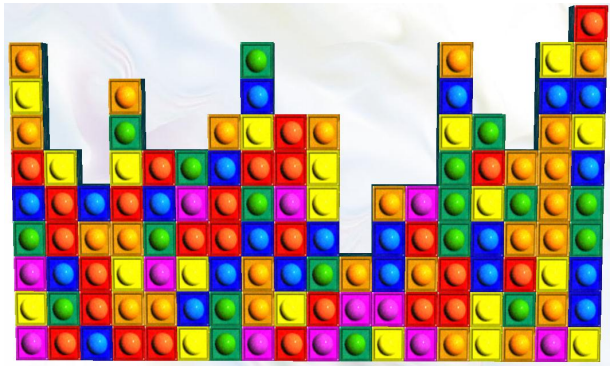

Figure 1: A Clickomania instance [4].

loop in the grid such that all of the numbered squares are inside the loop. In addition, for each of the numbered squares, the number of squares inside the closed loop, that are either horizontally or vertically adjacent to it, must equal that number. See Figure 2 for an example. Deciding the solvability of Corral Puzzle has been shown to be NP-Complete by Friedman (2002a). This seems to be the only information about Corral Puzzle.

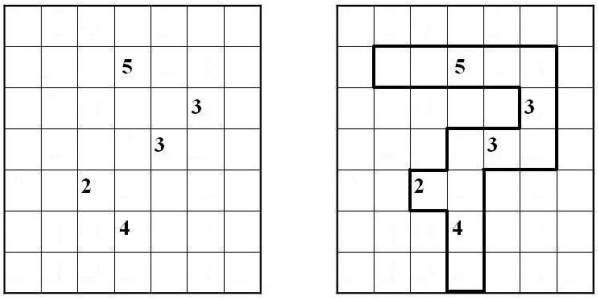

Figure 2: A Corral Puzzle instance and solution.

DECISION QUESTION (Friedman, 2002a): Does a given instance have a solution, that is, is it solvable?

(4) Cross Sum (also known as Kakuro) is a number puzzle played on an $m \times n$ grid of black and white squares. White squares form horizontal and vertical lines of two or more adjoining squares. Each horizontal and vertical line is labelled with a number representing its sum. The goal is to place a digit between $1 \ldots 9$ in every white box so that the sum of every line is equal to the label of that line, and so that no number is repeated for any one sum. See Figure 3 for an example. Deciding the solvability of Cross Sum has been shown to be NP-Complete by Takahiro (2001). Takahiro further defines (N, l, L)-CROSS SUM as the same problem except that every line of white has n boxes such that $l \leq n \leq L$, and each digit placed lies between $1 \ldots N$. Then $(9, 2, 6)$-CROSS SUM is NP-Complete, $(N, 2, 5)$-CROSS SUM (with $7 \leq N < \infty$) is NP-Complete, $(N, 1, 3)$-CROSS SUM (with $7 \leq N < \infty$) is NP-Complete, $(N, l, 2)$-CROSS SUM is linearly solvable, and $(2, l, L)$-CROSS SUM is also linearly solvable. See Takahiro (2001) for further information. We cannot find any other references to Cross Sum. For more information see http://www.pro.or.jp/~fuji/java/puzzle/crosssum/index-eng.html, which details some combinations of numbers that can be used to help find a solution, and lists some example problems and their solutions.

DECISION QUESTION (Takahiro, 2001) (also see Yato and Seta, 2003): Does the given instance have any configuration of numbers in the white boxes such that (1) the numbers in the white boxes are 1..9, (2) the sum of numbers in any row/column is equal to the corresponding number in the black box, and (3) a number only appears once in any given row/column?

(5) Cryptarithms (also known as Alphametics) is a number puzzle. An alphabet is used to encode some numbers of base k. The goal is to form a one-to-one function between the alphabet and the numbers so that the decoded mathematical formula is satisfied. The size of the alphabet is bounded by k. Numbers cannot have leading zeros. A famous example of Cryptarithms (with k=10) along with its solution is

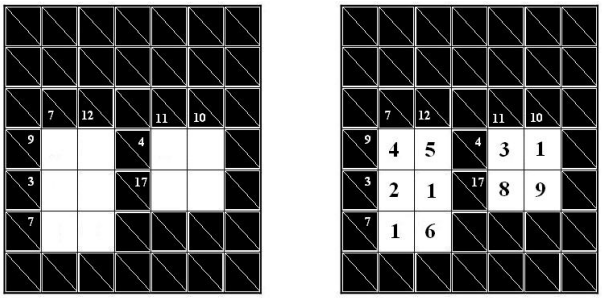

Figure 3: A Cross Sum instance and solution.

SEND	9567
+MORE	+1085
MONEY	10652

Deciding the solvability of a Cryptarithms problem has been shown to be NP-Complete by Eppstein (1987). There seems to be no other published work on Cryptarithms. The interested reader can find more information at http://www.geocities.com/Athens/Agora/2160/. This site provides a large number of example problems, some of which can be solved online, a tutorial on solving Cryptarithms, recommendations of books, and links to other websites. Also, http://bach.istc.kobe-u.ac.jp/llp/crypt.html provides downloadable source code for a solver.

DECISION QUESTION (Eppstein, 1987): Does the given instance have any solutions?

(6) Instant Insanity is played using cubes, which can be freely rearranged and rotated. The faces of each cube are coloured, such that with n cubes there are n unique colors. Note that having each cube of one colour is not allowed. The goal is to arrange the cubes into a single-file row, such that there are no repeated colors displayed along the top, front, bottom, and back of the row. Instant Insanity (with n=4) was a popular game by Parker Brothers. Deciding the solvability of Instant Insanity has been shown to be NP-Complete by Robertson and Munro (1978). Webster[5] offers an approach to Instant Insanity (n=4) that uses a graph to represent the problem and Walsh and Julstrom (1998) attempt to tackle it using a genetic algorithm (Sastry, Goldberg, and Kendall, 2005). More information can be found at http://www.cs.uidaho.edu/~casey931/puzzle/insane/insane.html, which includes the ability to print out a set of cubes that can be constructed in order to play the game.

DECISION QUESTION (Robertson and Munro, 1978): Given *n* cubes, with the faces of each cube coloured with one of *n* colours, is it possible to stack the cubes so that each colour appears exactly once on each of the 4 sides of the stack?

(7) KPlumber is a computer puzzle game played on an $m \times n$ grid of tiles. Each tile in the interior of the grid has four adjoining tiles, the corner tiles have two adjoining tiles, and the edge tiles have three adjoining tiles. At the centre of each tile is an intersection of up to four pipes, each of which runs directly to one of the four sides of the tile. Some tiles have no pipes. Water runs through the pipes and leaks from any that are left open. An open pipe at the edge of a tile that touches an open pipe at the edge of the adjacent tile closes both of the pipes. An action rotates one tile, and the pipes on it, by 90°. The goal is to arrive at a situation in which all pipes are closed so that there is no leaking water. This is known as a *safe state*. Deciding the solvability of KPlumber has been shown to be NP-Complete by Kral *et al.* (2004), who also show that some restrictions as to the types of tiles that are allowed results in polynomial-time decidability. KPlumber has not received any other attention, as far as the authors are aware. The interested reader can visit http://www.vanderlee.com/software_linkz.html, which provides a freeware download of a game based on KPlumber called Linkz. Figure 4 shows a screen shot of an initial KPlumber puzzle and the same puzzle in a solved state.

DECISION QUESTION (Kral *et al.*, 2004): Given an instance of KPlumber, is it possible to rotate the tiles so as to reach a safe state? Note: There are some definitions of the problem, which are not NP-Complete (see Kral *et al.*, 2004)

(8) Lemmings is a popular computer game from the early 1990s. The rules of the game are complex, therefore this description shall be brief and not definitive. We refer to Spoerer (2007, Ch. 5) for a definition of LEMMINGS, which is a derivative of Lemmings. Lemmings is played over a two-dimensional display of lemming entities

[5] See http://www.csulb.edu/~fnewberg/PCTMSummary/FinalPDFs/francine.PDF.

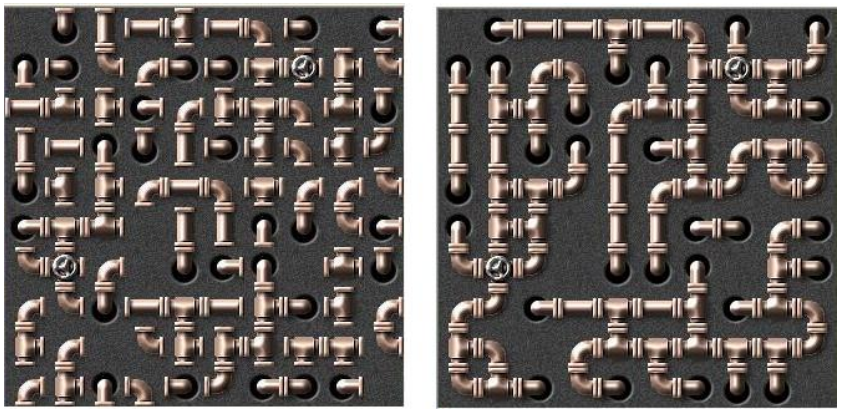

Figure 4: A KPlumber instance and solution (http://www.vanderlee.com/software_linkz.html).

that move around an environment. The player interacts with the lemmings. Each of the lemmings enter the environment at some specified position and must be guided to an exit. Each one that exits is counted. A lemming will continue to walk until either it is blocked, whereby it turns around, or until it falls down a hole. An action assigns a type to a single lemming. There are eight lemming types

1. **Climber**. This makes the lemming climb up all vertical walls that it comes into contact with.
2. **Floater**. This makes the lemming float slowly down instead of falling, thus stopping it from dying on impact with the ground.
3. **Bomber**. This makes the lemming explode after a countdown. The explosion kills the lemming and destroys the peripheral environment, but it does not affect other lemmings.
4. **Blocker**. This makes the lemming stand still and block the movement of the other lemmings, so that they reverse direction.
5. **Builder**. This makes the lemming create a bridge.
6. **Basher**. This makes the lemming bash a horizontal tunnel through a solid wall.
7. **Miner**. This makes the lemming mine a diagonal tunnel downwards through a solid wall.
8. **Digger**. This makes the lemming dig a vertical tunnel downwards through the solid platform.

Each lemming type has a specified quantity, such that the number of times that a type is used during a play does not exceed its quantity. The goal is to guide a specified number, m, of the n lemmings through the environment to an exit within a specified time limit. Deciding the solvability of Lemmings has been shown to be NP-Complete by Cormode (2004), who also shows that deciding the solvability of a restriction to $m = n = 1$ is still NP-Complete, and further that a restriction to climber and floater types only is decidable in P time.

John McCarthy (1998) offers Lemmings as a new "*Drosophila for AI research, connecting logical formalizations with information that is incompletely formalizable in practice*", and goes on to discuss the features of Lemmings that make it a challenge to both experimental and theoretical AI. Recall that John McCarthy originally named the AI discipline at the Dartmouth conference in 1956 (see McCarthy *et al.* (1955)). The game of Chess has traditionally been cited as the Drosophila of AI research. That is, any insight gained in AI for the game of Chess might provide researchers with some understanding in AI for other games or maybe some other types of problems. Chess received decades of attention from researchers until eventually DEEP BLUE (Campbell, Joseph Hoane Jr, and Hsu, 2002) defeated the World Chess Champion, Garry Kasparov, in 1997. Research in Chess is still very active (see also publications elsewhere, Fogel *et al.* (2006), Sadikov and Bratko (2006), Hallam, Poh, and Kendall (2006), Kendall and Whitwell (2001))[6]. However, it no longer attracts the media attention it did in the mid 1990s. McCarthy advocates the use of Lemmings as a model for Logical AI and also suggests that other approaches might benefit from using Lemmings as a test domain. McCarthy suggests approaching Lemmings

[6]Of course, this journal provides a regular source of research for Chess.

using Situation Calculus (see Russell and Norvig, 2003, p. 329). He further highlights the concurrent nature of the game that arises because there are many lemmings with which the player interacts and which are integral to the game. Kendall and Spoerer (2004) approach a sub-problem of a derivative of Lemmings using a genetic algorithm. More information about Lemmings can be found at http://lemmings.deinonych.com/, which offers information about the various Lemmings games and links to other sites.

DECISION QUESTION (Cormode, 2004): Given a level of Lemmings, is there a winning strategy?

(9) Light Up is played on an $m \times n$ grid of black and white squares. An action (clicking on a cell) places a light bulb on a white square. Some of the black squares have a number between 0 and 4 that represents the number of light bulbs that should be placed directly next to that square, horizontally or vertically. A light bulb illuminates the row and column of white squares in each of the four directions, until the $beam$ reaches a black square or the edge of the grid. The goal is to illuminate every single white square, without any light bulb illuminating another light bulb. Deciding the solvability of Light Up has been shown to be NP-Complete by McPhail (2005). Light Up has not received any other attention, at least as far as the authors are aware, except for a recent article which utilized a Hopfield neural network and a genetic algorithm (Ortiz-Garcia *et al.*, 2007). More information can be found at http://www.puzzle-light-up.com/, which provides a set of sample problems. Figure 5 shows a completed Light Up puzzle.

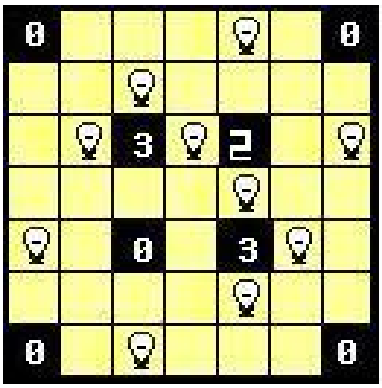

Figure 5: A completed Light Up Puzzle (http://www.puzzle-light-up.com/).

DECISION QUESTION (McPhail, 2005): Given a Light Up puzzle, does a solution exist?

(10) Mastermind is a popular game, which has been sold as a two player game where one person sets the problem and the other player tries to find the solution. However, it can easily be converted to a single player game, with the computer setting the puzzle and providing the relevant feedback.

The aim of the game is to find the correct colour *and* sequence of four pegs (see Figure 6). The computer selects four pegs and arranges them. The player then has eight guesses to find the correct sequence/colour of the hidden pegs. The player makes a guess by arranging four pegs. The computer responds by showing a black peg for any peg which is the correct colour *and* is in the correct position. A white peg indicates that a peg is of the correct colour but is not in the correct position. As the game progresses, the player can use information from previous guesses to influence their next action. Stuckman and Zhang (2006) has shown that a generalized Mastermind puzzle is NP-Complete. There has also been some other work on solving mastermind problems. See, for example, Neuwirth (1982), Chvátal (1983), Bernier *et al.* (1996), Kalisker and Camens (2003).

DECISION QUESTION (Stuckman and Zhang, 2006): Given a set of guesses, and their corresponding scores, is there at least one valid solution?

(11) Minesweeper is a popular computer puzzle game that comes with some Microsoft™ operating systems. It is played on an $m \times n$ grid of cells, all of which are initially hidden. k mines are randomly distributed over the grid. Clicking on a cell reveals its contents. If the cell contains a mine then the game ends and the player loses. If there is not a mine in the cell, a number is revealed (1..8) that corresponds to the number of adjacent (horizontally, vertically or diagonally) cells that contain a mine. If a cell does not contain a mine, and is not next to a mine, a blank cell is displayed and other adjacent blank cells are also revealed. The game ends when, either a mine is uncovered (a loss) or all cells are revealed that do not contain a mine (a win). Figure 7 shows an end state of a Minesweeper puzzle. In this case it is a loss for the player as one of the mines (three rows up) was revealed.

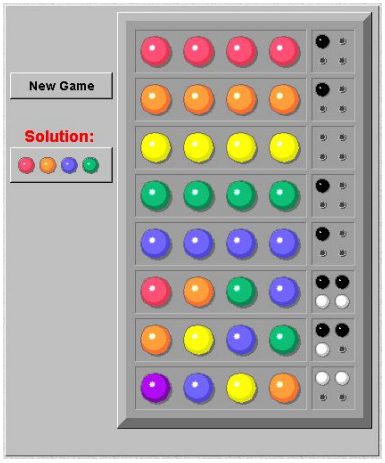

Figure 6: A completed Mastermind Puzzle (http://nlvm.usu.edu/en/nav/frames_asid_179_g_2_t_1.html).

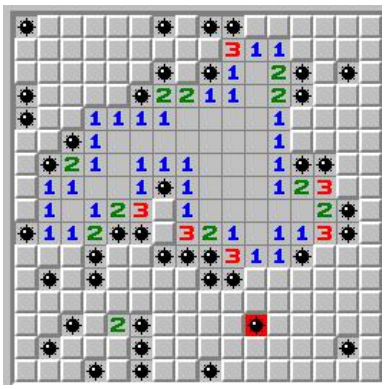

Figure 7: Final Position of a Minesweeper Puzzle (Microsoft Operating System).

The goal is to reveal all of the cells that do not contain a mine, leaving the k cells that do contain a mine still hidden. Deciding the solvability of Minesweeper has been shown to be NP-Complete by Kaye (2000). We can define l as the maximum number of mines that surround any cell, and then general Minesweeper has $l = 8$. McPhail (2003) shows that deciding the solvability of Minesweeper with $l = 3$ is also NP-Complete. Castillo and Wrobel (2003) describe a system to learn Minesweeper playing strategies. Adamatzky (1997) applies Minesweeper to Cellular Automaton. Also see Quartetti (1998), Rhee (2000) for evolutionary approaches to Minesweeper.

DECISION QUESTION (Kaye, 2000): Given an arbitrary set of mines and numbers on a rectangular grid, can mines be placed consistently, following the usual minesweeper rules?

(12) n-Puzzle. Many believe that this puzzle was invented by Sam Lloyd in the 1870s (Lloyd, 1959). In fact, Noyes Palmer Chapman is now credited with its invention as shown in Slocum and Sonneveld (2006), who document the puzzle's history. The puzzle made its first appearance in the scientific literature in 1879 (Johnson and Storey, 1879). The 8-puzzle problem was later discussed in Doran and Michie (1966). The game is played on an $m \times m$ grid (the puzzle can also be played on an $m \times p$ grid but, for simplicity, we restrict the discussion to $m \times m$ grids), such that $m^2 - 1 = n$. Within the grid are n square tiles, each of which has a unique number from $1 \ldots n$. Thus, one square in the grid is always empty, called the 'blank' and, depending on its position, the blank has two, three, or four adjacent tiles. A move puts one of these tiles into the blank and relocates the blank to that tile's previous position. This is the same as the blank moving, and some solution methodologies model the problem in this way. The tiles are initially set to some random configuration. The goal is to rearrange the tiles into a configuration such as that shown in Figure 8 for the 8-Puzzle. Korf (1999) gives an excellent overview of the n-puzzle problem, with particular reference to the 8-, 15- and 24-puzzle. It is interesting to note that the game divides into two state sets such that it is not possible to move from a state in one set to a state in the other state set; using just legal moves. The 8-puzzle is easy to solve as the search space is relatively small (181,440),

although A* with a suitable heuristic (for example the Manhattan distance) improves the solution procedure. The 15-puzzle cannot be solved with A* but a refinement (Korf, 1985), known as the IDA* algorithm, was the first to obtain optimal solutions to this variant. More recently, Korf and Schultze (2005) provided an improvement to best first search that allowed them to complete the first breadth-first search of the 15-puzzle problem, which has over 10^{13} states. Optimal solutions to the 24-puzzle cannot be generated in reasonable times using IDA* with the Manhattan heuristic and it requires better heuristics such as those proposed by Hannsson, Mayer, and Yung (1992). The interested reader is recommended to read Korf (1985, 1999), Korf and Felner (2002), as these articles are an excellent source of information, which we would only repeat here. Finding a solution to n-Puzzle with the fewest number of moves has been shown to be NP-Complete by Ratner and Warmuth (1990). Reinefeld (1993) provides details of experiments on 8-Puzzle with IDA* and operator selection. Culberson and Schaeffer (1996) apply pattern databases to the 15-puzzle, and Korf and Taylor (1996) perform experiments on 24-Puzzle with IDA*. Parberry (1995) reports an algorithm for finding sub-optimal solutions in polynomial time. Taylor and Korf (1997) detail a technique for pruning duplicate nodes from a search and test it on 15-Puzzle and 24-Puzzle. In Korf and Felner (2002), the authors present the current state of the art for producing optimal solutions for these problems, using ideas based on IDA* with pattern database heuristics.

The interested reader can find more information at http://www-cse.uta.edu/~cook/ai1/lectures/applets/ep/ep.html, which allows the user to play 8-Puzzle, and to also solve 8-Puzzle problems. Slocum and Sonneveld (2006) is also an excellent reference, which should be regarded as essential reading.

2	8	3
1		5
4	7	6

1	2	3
4	5	6
7	8	

Figure 8: An 8-puzzle instance and goal state.

DECISION QUESTION (Ratner and Warmuth, 1990): Is there a solution for transforming an initial configuration into a goal configuration requiring at most k moves? The initial state and goal state are defined as $n \times n$ board configurations. This proof applies to the general $(n^2 - 1)$-puzzle.

(13) Nurikabe is played on an $m \times n$ grid of white squares. An action shades one of the squares. Some of the squares contain a number and cannot be shaded. The goal is to shade the grid such that each group of white squares contains exactly one number that represents the number of squares in the group. In addition, there cannot be any 2×2 blocks of shaded or unshaded blocks of squares. See Figure 9 for an example. Deciding the solvability of Nurikabe has been shown to be NP-Complete by McPhail (2003). Nurikabe has not received any other attention, at least as far as the authors are aware. See http://www.puzzle.jp/letsplay/play_nurikabe-e.html for a set of sample problems.

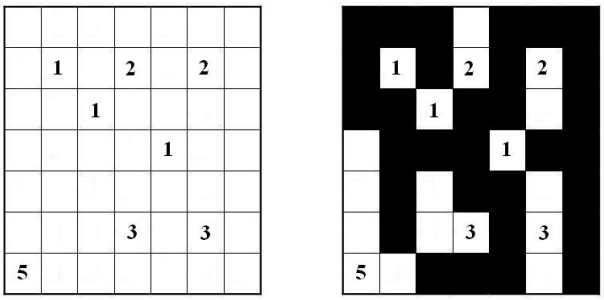

Figure 9: A Nurikabe instance and solution.

DECISION QUESTION (McPhail, 2003): Given an $n \times m$ instance *I* of Nurikabe, does there exist a partition that satisfies the following conditions?

1. Each partition of cells is contiguous.
2. No numbered cell can be shaded.
3. There cannot be any 2×2 blocks.

4. Each contiguous block of white cells only contains one numbered cell and that each of these blocks contains a number of cells equal to the value given in the numbered white square within that block.

A more formal definition is given in McPhail (2003).

(14) Pearl Puzzle is played on an $m \times n$ grid of squares. Each square can contain a white pearl, a black pearl, or be empty. The goal is to construct a closed path that does not cross itself, and that passes through every pearl such that the path turns 90° through each black pearl and is straight immediately before and afterwards, and such that the path is straight through every white pearl and turns 90° immediately before or afterwards. See Figure 10 for an example. Deciding the solvability of Pearl Puzzle has been shown to be NP-Complete by Friedman (2002b) and this appears to be the only information about Pearl Puzzle.

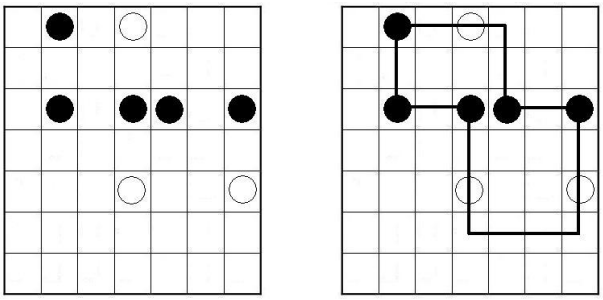

Figure 10: A Pearl Puzzle instance and solution.

DECISION QUESTION (Friedman, 2002b): Is there a closed, non-intersecting path passing through every pearl so that:

1. the path turns at every black pearl, but does not turn immediately before or after, and
2. the path does not turn at any white pearl, but does turn immediately before or after.

(15) Peg Solitaire (also known as Hi-Q) is played on an $n \times n$ grid of holes, typically arranged in the shape of a cross (although other shapes can be used). A peg is usually placed in every hole, except the centre one. A move takes a peg, and jumps it over another peg, that is horizontally or vertically adjacent, and places it in an empty hole behind it. The peg that was jumped over is removed. The usual goal is to leave a single peg remaining in the centre hole but other goal states can also be defined. There are many other variations, besides just the arrangement of holes, the starting configuration and the goal state. For example, Reverse Peg Solitaire adds pegs to the board by jumping over empty holes and placing a peg in the hole that was jumped. The aim, in this variant, is to arrive at some given configuration of pegs. Deciding the solvability of Peg Solitaire has been shown to be NP-Complete by Uehara and Iwata (1990). Moore and Eppstein (2002) show that the restriction to a single line of pegs is solvable in polynomial time, and Ravikumar (2004) replaces this by showing that Peg Solitaire played on a $k \times n$ grid, for any constant k, is solvable in linear time. Kiyomi and Matsui (2000) apply integer programming algorithms to Peg Solitaire, and Matos (1998) applies Depth First Search to the puzzle. Also see Jefferson *et al.* (2006) for details of various models of Peg Solitaire. The interested reader can find more information in Beasley (1992) and also at http://www.myparentime.com/games/games8/games8.shtml, where they can play Peg Solitaire and see a solution.

DECISION QUESTION (Uehara and Iwata, 1990): Given a generalized Hi-Q problem, is there a sequence of moves that leaves the final peg in the goal position? The generalized Hi-Q problem is defined as an initial position on an extended board of size $n \times n$. The goal is given as a position on the board where the final peg will be placed.

(16) Reflections is played on an $m \times n$ grid. There is a laser that emits a beam of light, a number of light bulbs, each of which must be hit by the beam an odd number of times and a number of bombs that explode if any beam hits them. There are immovable walls that block beams, and movable objects that interact with the beam in a variety of ways. For example, mirrors that change the direction of the beam by 90°, refractors that divert the beam by 45°, and splitters that split the beam into two. An action moves or rotates one of the objects. Deciding the solvability of Reflections has been shown to be NP-Complete by Kempe (2003). There appears to be no other

work on Reflections. More information can be found at http://laser.narr.as/, which has a version of the game. Figure 11 shows examples of two initial states. The left hand figure has three mirrors and is relatively easy to solve. The right hand figure shows a slightly more complex puzzle with two mirrors and one refractor. There are also one way devices on the playing area in the right hand figure. These only allow the beam to pass in one direction and these objects cannot be rotated or moved.

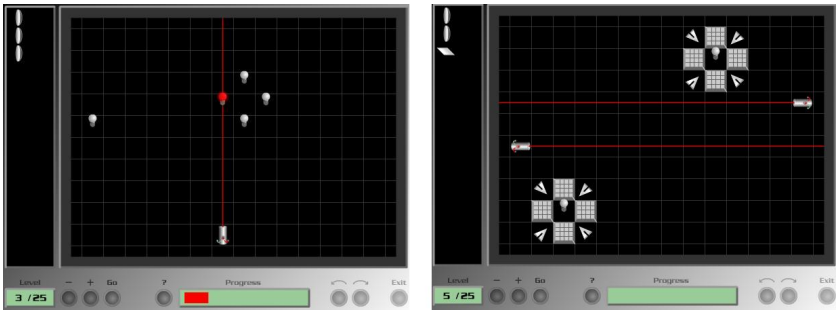

Figure 11: Two initial states of Reflections (http://laser.narr.as/).

DECISION QUESTION (Kempe, 2003): Can the items (or a subset of) be placed on the grid such that all the lamps are hit by an odd number of light rays, and no bomb is hit by a light ray?

(17) Rush Hour was created by Nob Yoshigahara. The game is played on a 6 x 6 grid, on which there are a number of cars; one of them being the car we have to manoeuvre out of the grid. The cars occupy two or three squares and can only move forwards or backwards (but not sideways) (see Figure 12) The aim is to move the cars in such a way that the red car can be driven out of the exit (see Figure 12 for a solution state). The generalized rush hour problem, which has an arbitrary grid size and allows the exit to be at any point on the perimeter of the grid, has been shown to be PSPACE-complete by Flake and Baum (2002). As this is a higher level of complexity than NP-Complete we believed that the game is worth to be listed in this survey. Other work on this puzzle is by Servais (2005) and Fernau *et al.* (2003).

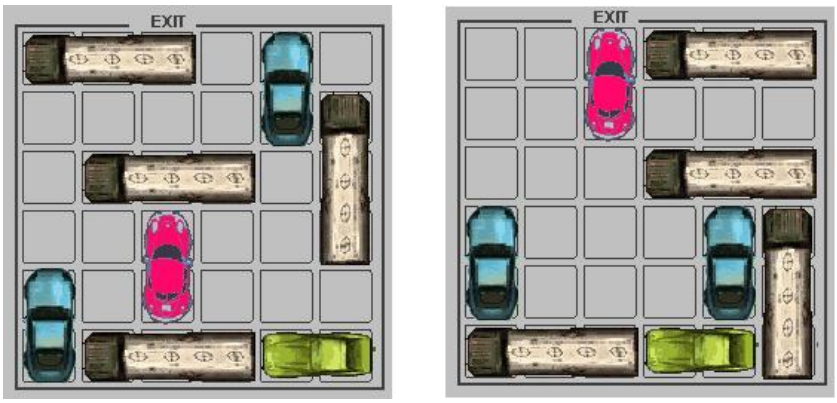

Figure 12: A Rush Hour instance and solution (www.puzzles.com/products/RushHour/RushHourApp.htm).

DECISION QUESTION (Flake and Baum, 2002): Does a solution exist?

(18) Shanghai (also known as Mahjong Solitaire) is played using 144 Mahjong tiles. They are laid out in various patterns, with some tiles being placed on top of others. Tiles are removed by selecting matching tiles, which removes those two tiles from play. The objective is to remove all the tiles. Only tiles which are *in play* can be selected. If a tile has two neighbouring cells, at the same level, on either side of it, it cannot be selected. Deciding the solvability of Shanghai, where the position of every tile is known, has been shown to be NP-Complete[7]. See http://www.gamegarage.co.uk/info/shanghai-mahjong/ in order to play a trial version of the puzzle.

DECISION QUESTION[7]: Is it possible to remove all matching pairs of uncovered tiles until all tiles have been removed, even if all tile positions are known?

[7] See http://www.ics.uci.edu/~eppstein/cgt/hard.html#shang.

(19) Slither Link is played on an $m \times n$ grid, with dots marking the intersections. Some of the cells are numbered between 0 and 3. An action connects two adjacent dots with a vertical or horizontal line. The goal is to form a single loop in the grid of dots that does not cross or branch. The number in each cell represents the number of lines that surround that cell. An empty cell can be surrounded by any number of lines. See Figure 13 for an example. Slither Link has been shown to be NP-Complete by Yato (2000)[8]. Slither Link has received no other research attention that we are aware of. The interested reader can attempt the puzzle at http://www.puzzle-loop.com/ and http://www.pro.or.jp/~fuji/java/puzzle/numline/index-eng.html.

DECISION QUESTION (Yato, 2000)[8] (also see Yato and Seta (2003)): Given a slither link instance, does it have any solutions?

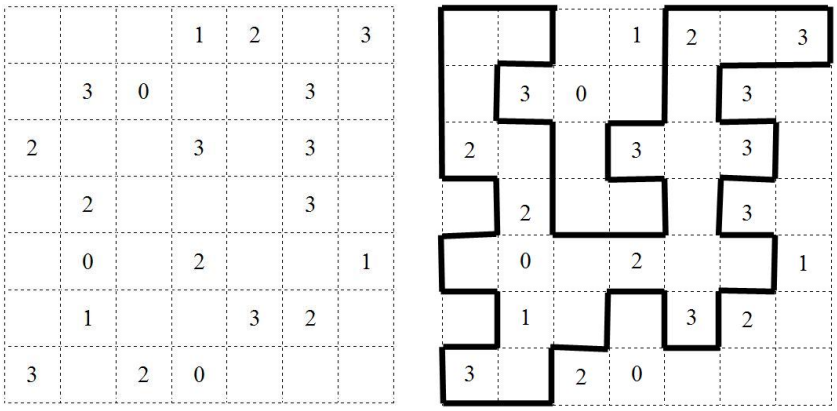

Figure 13: A Slither Link instance and solution.

(20) Sokoban is played on an $m \times n$ grid. Each cell is solid or empty, and empty cells can contain a stone or a target, such that there are k stones and k targets. In the grid is a moving device. An action moves the device one cell horizontally or vertically as long as there is no solid cell blocking the way. Moving the device into the cell occupied by a stone will also move that stone one cell in the same direction, provided that the stone is not blocked by a solid cell or another stone. The goal is to move each of the stones to an empty target. Deciding the solvability of Sokoban has been shown to be NP-Complete by Dor and Zwick (1999). Junghanns and Schaeffer have performed various investigations on Sokoban. In Junghanns and Schaeffer (1998) they apply IDA* (Korf, 1985) in order to identify Sokoban states that provably cannot be solved. In Junghanns and Schaefer (1999, 2001) they prune moves from the search that are not deemed to be relevant to the recent sequence of moves. Murase, Matsubara, and Hiraga (1996) describe a Sokoban problem generator. See http://www.cs.ualberta.ca/~games/Sokoban/ for information relating to the history of Sokoban, a list of publications and a description of the Rolling Stone Sokoban solver. Moreover, we refer to Junghanns' (1999) PhD thesis that contains references to other work on Sokoban. Sokoban is included in a larger set of problems that are known as motion planning problems (O'Rourke and The Smith Problem Solving Group, 1999) and pushing block problems (Demaine, 2001). In general there is a single moving device that can push (or pull) some number of objects simultaneously, of some predefined shape, which have inertia or not, in order either to arrange those objects or to move to a goal position, in 2 or 3-dimensions. These problems have generally been shown to be NP-Hard. We refer to O'Rourke and The Smith Problem Solving Group (1999) and Demaine (2001) for more information about these other problems.

DECISION QUESTION (Dor and Zwick, 1999) (also see Papadimitriou *et al.* (1994), who address a slightly different problem): Given a Sokoban instance, is there a solution?

(21) Solitaire (Card Games) are usually played with a single deck of cards and, as the name suggests, they are played by a single player. The usual objective is to *get out*, where the definition of *getting out* depends on the rules for the particular variant being played. It is impossible to list every solitaire variant as, not only are there so many, but each variant has many different rules. See, for example Morehead and Mott-Smith (2001).

FreeCell is a popular solitaire game, largely due to the fact that it is included with Windows™ installations, which is why we decide to describe it here. It is played with a standard deck of 52 cards. These are initially arranged into eight stacks of six or seven cards each, with every card being visible (see Figure 14). Cards can

[8] In Japanese

be moved between the stacks, the four free cells (top left in Figure 14) and the four foundation cells (top right in Figure 14). The cards are subject to the following rules.

1. Only cards in a free cell or on the top of a stack may be picked up.
2. Only one card can be picked up at any time.
3. A card may be added to a non-empty stack if the card being deposited is one less than the current value and has a different colour. For example, placing 3♠ on 4♢.
4. Cards may be stored in a free cell, but only one card can occupy any free cell.
5. If a stack is empty, a card may be added to it (making it a stack with one card).
6. Aces may be placed on the empty foundation stacks. Following this, cards may be added to the foundation stacks as long as they are one value higher and of the same suit. For example, placing 2♣ on A♣.

This description is based on the original rules. On our Windows™ version the game plays slightly differently. For example, we can pick up more than one card from a stack when there are consecutive numbers and different colors. The aim of the game is to move all the cards to the foundation stacks.

Helmert (2003) has shown that FreeCell is NP-Complete. The proof is presented in the context of a planning domain and FreeCell is used as a final example in the paper. As such, it is not possible to give the decision problem here; the interested reader is referred to the article.

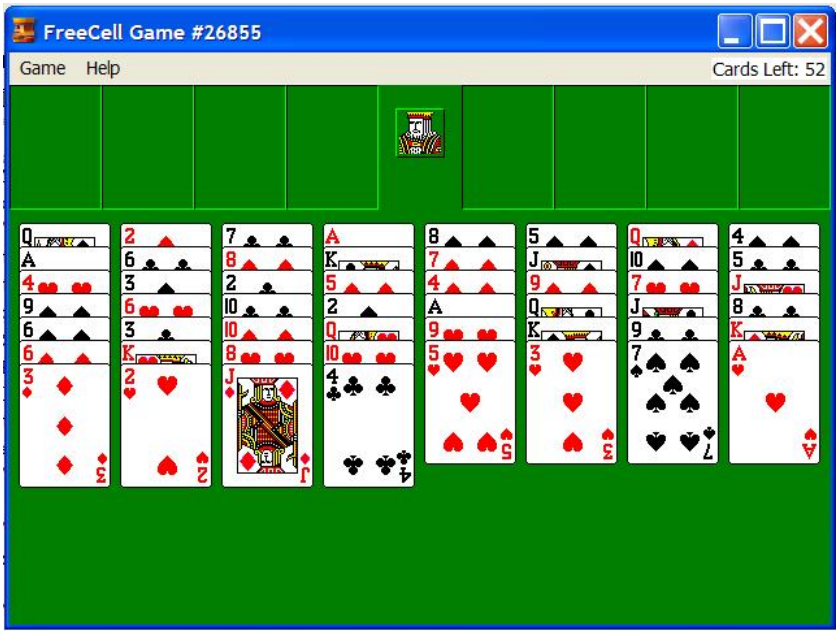

Figure 14: FreeCell: Solitaire Card Game (Windows™ XP).

(22) Spiral Galaxies is played on an $m \times n$ grid. k markers are within the grid, located in the centre of cells, on the partitions between cells or at the intersection of four cells. The goal is to separate the grid into k segments so that each segment contains a marker at its centre, and each segment is symmetrical under 180° rotations. Deciding the solvability of Spiral Galaxies has been shown to be NP-Complete[9]. It seems to be the only work on this puzzle. Figure 15 shows a screen shot of an initial Spiral Galaxies puzzle and the same puzzle in a solved state.

DECISION QUESTION[9]: Given an instance of Spiral Galaxies, is there a solution?

(23) Sudoku (also known as Number Place) was first published in Dell Magazines in 1979[10]. It is currently attracting more attention than other puzzles, perhaps because it is featured regularly in daily news articles. It is

[9] http://www.stetson.edu/~efriedma/articles/spiral.pdf
[10] See http://en.wikipedia.org/wiki/Sudoku.

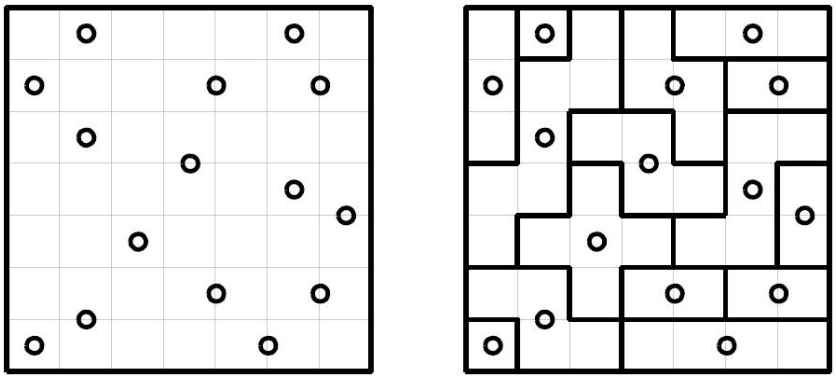

Figure 15: A Spiral Galaxies instance and solution[9].

played on an $n^2 \times n^2$ grid that is divided into $n \times n$ equal segments. Initially some cells in the grid contain a number between $1 \ldots n^2$. An action inserts a number between $1 \ldots n^2$ into an empty cell. The goal is to fill the grid with numbers such that every row, column, and segment contains the numbers $1 \ldots n^2$ without any repetitions. Deciding the solvability of Sudoku has been shown to be NP-Complete by Yato and Seta (2003) (as is the closely related problem of constructing Latin squares from partially filled squares (Colbourn, Colbourn, and Stinson, 1984)). Rodrigues Pereira, de Castro Dutra, and Clicia Stelling de Castro (2001) apply constraint programming techniques to Sudoku. Nicolau and Ryan (2006) offer an evolutionary approach to Sudoku. We refer to Eppstein (2005), Lynas and Stoddart (2005), and Lewis (2007) for other approaches to Sudoku. The world wide web is littered with Sudoku solvers. See http://www.sudoku.org.uk/ for some sample problems that can be attempted online.

DECISION QUESTION (Yato and Seta, 2003): Given a Sudoku instance, does it have any solutions?

(24) Tetris is a popular computer puzzle game that is played on an $m \times n$ grid. Each cell is either empty or contains a solid. The game is played with pieces that are made up of combinations of four squares of solid. The pieces fall downwards from the top of the grid until they either hit the bottom of the grid or hit another solid. A row of solids is deleted from the grid when it is formed, and everything above falls downwards. A tetris is the simultaneous deletion of four consecutive rows. An action moves the falling piece once to the left or right or rotates the piece. Demaine, Hohenberger, and Lieben-Nowell (2003) and Breukelaar *et al.* (2004) have reasoned about the offline version of Tetris. That is, where the sequence of pieces that will drop from the top of the grid are included as part of the problem instance. They show that the following are all NP-Complete - maximizing the number of deleted rows, maximizing the number of tetrises, maximizing the number of pieces that are placed, or minimizing the maximum height of a solid cell. In Hoogeboom and Kosters (2004) another version of Tetris is also shown to be NP-Complete. Hoogeboom and Kosters (2005) present a summary of the NP-Completeness proofs for Tetris. Siegel and Chaffee (1996) and Yurovitsky (1995) use Tetris to test an evolutionary algorithm. Germundsson (1991) reports some work on a variation of Tetris. For more information see http://www.ebaumsworld.com/tetris.html in order to play a version of Tetris.

DECISION QUESTION (Demaine *et al.*, 2003; Breukelaar *et al.*, 2004): Given a Tetris game, is there a *trajectory sequence* so that we can clear *k* rows without incurring a loss? A trajectory of a given piece is defined as a sequence of legal moves (that is, its initial position, the moves it makes (downward moves, sideways moves and rotations, and its final position). A trajectory sequence consists of the combined trajectories of all the playing pieces. Other variations are also considered in Demaine *et al.* (2003). These include the following questions.

1. Is there a trajectory sequence that contains at least *k* tetrises (when four rows are cleared at the same time)?
2. Can *n* pieces be placed before a loss occurs?
3. Does the trajectory sequence never fill the grid square above height *h*?

We would also refer the reader to Hoogeboom and Kosters (2004) for different tetris variants and NP-Completeness proofs.

4. TWO OTHER PUZZLES

In listing the 24 puzzles above, we recognize that there may be some omissions. Some of the puzzles we have missed will be due to the fact that we are simply unaware of the problem, or the existence of a proof that the puzzle is NP-Complete. Other puzzles have been purposefully omitted, for various reasons. For example, **Rubik's Cube** would seem to be an obvious omission. We describe it briefly here, due to the interest the puzzle holds.

(25) Rubik's Cube was invented in 1974, by Ernö Rubik. It started to gain popularity in the West when it was marketed by Pentangle Puzzles in 1978. This popularity was boosted by an article (Six Sided Magic) by David Singmaster in *The Observer* on the 17$^{\text{th}}$ June 1979. It was originally called The Magic Cube, being renamed in 1980 to Rubik's Cube. It is estimated that at least 300,000,000 cubes have been sold worldwide. The standard puzzle is played using a $3 \times 3 \times 3$ cube. Each of the nine exposed squares of the six faces has a coloured sticker on it. Six colours are used and when the cube is orientated correctly all the cubes on one face have the same colour. Each $3 \times 3 \times 1$ planes can be rotated independently to the rest of the cube. The cube is initially orientated so that each of the six faces have the same colour. Just a few moves disrupts this initial configuration so that is it not obvious how to get back to the starting position. The challenge is to manipulate the cube so that the original position is restored. There are many books and web pages that show how to solve Rubik's Cube. http://www.geocities.com/jaapsch/puzzles/cube3.htm, for example, provides many links and one solution method, while Frey and Singmaster (1997) and Singmaster (1981) provide a more theoretical treatment of the subject.

We are not aware of a reference to Rubik's Cube being NP-Complete, which is why we have not listed it in the main body of the article. However, there are many variants of the cube (for example, $4 \times 4 \times 4$, $5 \times 5 \times 5$, super cubing[11]), which are even more challenging than the standard version.

Like other puzzles, Rubik's Cube has been used as a test bed to demonstrate the effectiveness of various search techniques. Korf (1997) (reprinted in Van den Herik and Iida (2000), pp. 129-141) found the first optimal solutions to random instances of Rubik's Cube using Pattern Databases, showing that 18 moves were needed on average to solve a given instance. It is also interesting to note that Stephen Cook (1984) uses Rubik's Cube as an example that is easy to verify if a solution is correct, but it is difficult to find the solution given a randomly presented cube, but he does not provide a proof that Rubik's Cube is NP-Complete.

An excellent overview of Rubik's Cube can be found in Korf (1999).

(26) The Towers of Hanoi may seem a second obvious puzzle that we do not include in the above survey. Like Rubik's Cube, we are not aware of any work that shows that a generalization of this puzzle is NP-Complete. Of course, the puzzle has been intensively studied both from an algorithmic point of view (it is often used in programming courses to teach recursion) and from a complexity point of view as the steps needed to solve the problem grows exponentially with the number of pegs and the number of rings. Interested readers may want to refer to some recent work by Korf and Felner (2007), which provided a case study of the four pegs Towers of Hanoi Problem.

5. CONCLUSIONS

Despite trying to cover as many puzzles as possible we recognize that there will be some omissions, but we hope that the puzzles we have covered will promote more research attention in this area. In Table 1, we summarize the findings of this survey. We list each puzzle, the reference to its NP-Completeness proof and the main references to the work carried out. Of particular interest are the puzzles that have no references. This indicate that we have found no related work on the puzzle, which suggests that it could be a fruitful research area. In the remainder of Table 1, we have attempted to categorize the puzzles in two ways. The *Action Order Matters?* column details whether the order in which actions are carried out matters. The *Hidden Information?* column shows whether the complete state of the puzzle is known at all times. We have shown these last two columns in order to provide some initial analysis of the various puzzles we have reported. Future researchers might be able to establish some correlation between these measures and how the puzzles can be solved. Of course, there may be many other categorizations that can be drawn out. As potential starting points for other researchers, we also present some ideas below.

[11] Super Cubing is a challenge to return the cube back to the $exact$ original orientation (you can imagine that each face of the cube forms a picture which you must recreate).

Puzzle	NP-Completeness Proof	References	Action Order Matters?	Hidden Information?
Blocks World	Gupta and Nau (1992)	Slaney and Thiebaux (2001)	Y	N
Clickomania	Biedl *et al.* (2002)	Biedl *et al.* (2002)	Y	N
Corral Puzzle	Friedman (2002a)	-	N	N
Cross Sum	Takahiro (2001)	-	N	N
Cryptarithms	Eppstein (1987)	-	N	N
Instant Insanity	Robertson and Munro (1978)	Walsh and Julstrom (1998)	N	N
KPlumber	Kral *et al.* (2004)	-	N	N
Lemmings	Cormode (2004)	McCarthy (1998), Kendall and Spoerer (2004)	Y	? (we are uncertain)
Light Up	McPhail (2005)	Ortiz-Garcia *et al.* (2007)	N	N
Mastermind	Stuckman and Zhang (2006)	Kalisker and Camens (2003), Neuwirth (1982), Chvátal (1983), Bernier *et al.* (1996)	Y	Y
Minesweeper	Kaye (2000)	Castillo and Wrobel (2003), Adamatzky (1997), Quartetti (1998), Rhee (2000)	Y	Y
n**-Puzzle**	Ratner and Warmuth (1990)	Korf (1999), Korf (1985), Korf and Felner (2002)	Y	N
Nurikabe	McPhail (2003)	-	N	N
Pearl Puzzle	Friedman (2002b)	-	N	N
Peg Solitaire	Uehara and Iwata (1990)	Kiyomi and Matsui (2000), Matos (1998)	Y	N
Reflections	Kempe (2003)	-	N	N
Rush Hour	Flake and Baum (2002)[12]	Servais (2005), Fernau *et al.* (2003)	Y	N
Shanghai	See URL footnote[7]	-	Y	N
Slither Link	Yato (2000)	-	N	N
Sokoban	Dor and Zwick (1999)	Junghanns and Schaeffer (1998)	Y	N
Solitaire	Helmert (2003)	-	Y	Y
Spiral Galaxies	See URL footnote[9]	-	N	N
Sudoku	Yato and Seta (2003)	Rodrigues Pereira *et al.* (2001), Nicolau and Ryan (2006)	N	N
Tetris	Demaine *et al.* (2003)	Siegel and Chaffee (1996), Yurovitsky (1995)	Y	Y

Table 1: Summary of Puzzles.

- The properties of the puzzle, e.g., is it self-updating or are the updates at the discretion of the user?
- Is it a multi-goal or a single goal puzzle?
- Can we design algorithms that can produce realistic random instances and can we design specific instances that can be classified as easy and hard?
- For what puzzles can we design instances where the optimal solution is known so that we can compare algorithms by how close they get to the optimal solution?
- Related to the above, some puzzles are binary in that you solve them or not. Is this classification useful?
- Is the puzzle stochastic or deterministic?
- How large is the search space?
- What is the best known algorithm for a given puzzle and is this information useful when looking at other puzzles? For example, some puzzles might be amenable to being solved using constraint programming. An obvious example is Sudoku, as it can naturally be modelled by constraint programming, using an *all different* constraint. Other puzzles, such as Light Up and Nurikabe, might also be solvable, in practise, using a constraint programming formulation.
- Is deadlock possible and is this feature useful when analyzing different puzzles?
- In the introduction we defined a puzzle as "*A puzzle should have a solution which is aesthetically pleasing and gives the user satisfaction in reaching it.*" We recognized that this was very informal and we wonder if it is possible to produce a more formal definition, which would be generally accepted.
- In our definition, we said that the solution should be *aesthetically pleasing*. By this we mean that the user should recognize the intrinsic beauty of the solution. Whether, for example, this is the fact that they know they have adhered to all the constraints for Sudoku; the fact that *m* Lemmings have been saved or that all mines have been identified in Minesweeper. We believe that there is some research potential to investigate what gives the user satisfaction in solving a given puzzle, which might give insights into what would make a good new puzzle.
- This survey has looked at puzzles and attempted to find if they are NP-Complete. An obvious question is *Why not take an NP-Complete problem and transform it into a puzzle?* This raises a number of questions, not least of all, what makes a puzzle interesting (or even addictive)? For example, could the Travelling Salesman Problem make an interesting puzzle? Just because a problem is NP-Complete does not immediately make it suitable as a puzzle. Perhaps the instance(s) that are presented are too easy, or too hard. For a puzzle to be addictive, we suspect that the puzzle must be *averagely hard*. That is, neither too easy (so that the user quickly gets bored) or too hard (so that the user gives up). In addition, it should have different levels of difficulty to challenge the player as they make progress. Finally, the user must get some satisfaction from solving the puzzle.
- The puzzles in this survey have concentrated on those that are NP-Complete. Next to identifying puzzles missing from this survey and proving that other puzzles are also NP-Complete, it would be interesting to identify puzzles that are neither in P or NP-Complete. Graph isomorphism is an example of a problem that appears to have strange properties with regards to computational complexity. It is listed in Garey and Johnson (1979, p. 155) as being in NP, but not known to be in either P or NP-Complete, but rather a candidate for another class called NPI (NP problems of intermediate difficulty). It could be interesting to find puzzles with these types of properties as it might not only provide insight into the puzzle but may also add to our knowledge of computational complexity.

In carrying out this survey, we are aware that puzzles have been used as a research tool for many years, with valuable contributions by many leading scientists. However, there are still many challenges as to how they can best be solved (see the third column of Table 1). Advances in certain methods (such as IDA*) may be suitable for some puzzles, but other search (and planning) methodologies may be more suitable and new methods may need developing. The main aim of this article has been to collect many puzzles together so that the scientific community is aware of their existence.

[12] PSPACE-Complete Proof

We have described various NP-Complete puzzles and expanded Demaine's and Eppstein's surveys by discussing even more puzzles. We hope that this will motivate future research into these puzzles. We can analyze the puzzles in terms of the perceived amount of research that has been carried out to date. Table 1 summarizes our findings. The reason why some of these puzzles have received more research attention, whilst others have received very little, might be due to the accessibility of information about these puzzles. One motivation for this article is to bring many puzzles together in one place so that the reader has a starting point when they are looking for information about these puzzles, in particular the lesser known puzzles.

Acknowledgements

We are very grateful to the three anonymous referees for the detailed and helpful comments. The paper has been improved significantly and has been enriched by many details. We also thank Wendy Kendall for proof reading the article.

6. REFERENCES

Adamatzky, A. (1997). How cellular automaton plays Minesweeper. *Applied Mathematics and Computation*, Vol. 85, Nos. 2–3, pp. 127–137.

Beasley, J. D. (1992). *The Ins and Outs of Peg Solitaire (Recreations in Mathematics, No 3)*. Oxford University Press.

Bernier, J. L., Herriz, C. I., Merelo, J. J., Olmeda, S., and Prieto, A. (1996). Solving Master Mind using GAs and simulated annealing: A case of dynamic constraint optimization. *Parallel Problem Solving from Nature PPSN IV*, Vol. 1141 of *Lecture Notes in Computer Science*, pp. 554–563.

Biedl, T., Demaine, E., Demaine, M., Fleischer, R., Jacobsen, L., and Munro, J. (2002). The Complexity of Clickomania. *More Games of No Chance* (ed. R. Nowakowski), pp. 389–404, Cambridge University Press.

Breukelaar, R., Demaine, E. D., Hohenberger, S., Hoogeboom, H. J., Kosters, W. A., and Liben-Nowell, D. (2004). Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, Vol. 14, pp. 41–68. Special Issue: Selected Papers from the Ninth International Computing and Combinatorics Conference (COCOON 2003), Big Sky, MT, USA, July 2003.

Campbell, M., Joseph Hoane Jr, A., and Hsu, F. (2002). Deep Blue. *Artificial Intelligence*, Vol. 134, pp. 57–83.

Castillo, L. and Wrobel, S. (2003). Learning Minesweeper with Multirelational Learning. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 533–538.

Chvátal, V. (1983). Mastermind. *Combinatorica*, Vol. 3, Nos. 3–4, pp. 325–329.

Colbourn, C., Colbourn, M., and Stinson, D. (1984). *The computational complexity of recognising critical sets*, Vol. 1073 of *Lecture Notes in Mathematics*. Springer-Verlag, New York, NY.

Cook, S. (1971). The Complexity of Theorem Proving Procedures. *Proceedings of the third annual ACM symposium on Theory of Computing*, pp. 151–158.

Cook, S. A. (1984). Can Computers Routinely Discover Mathematical Proofs? *Proceedings of the American Philosophical Society*, Vol. 128, No. 1, pp. 40–43.

Cormode, G. (2004). The hardness of the lemmings game, or Oh no, more NP-completeness proofs. *Proceedings of the International Conference on Fun with Algorithms*, pp. 65–76.

Culberson, J. and Schaeffer, J. (1996). Searching with Pattern Databases. *Proceedings of the Biennial Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pp. 402–416, Springer-Verlag, London, UK.

Demaine, E. D., Hohenberger, S., and Lieben-Nowell, D. (2003). Tetris is Hard, Even to Approximate. *Computing and Combinatorics, 9th Annual International Conference*, Vol. 2697 of *Lecture Notes in Computer Science*, pp. 351–363, Springer-Verlag, Berlin, Germany.

Demaine, E. (2001). Playing Games with Algorithms: Algorithmic Combinatorial Game Theory. Technical report, Massachusetts Institute of Technology. http://arxiv.org/abs/cs.CC/0106019, last accessed 22 October 2006.

Devlin, K. (2005). *The Millennium Problems: The Seven Greatest Unsolved Mathematical Puzzles of Our Time*. Granta Books, London, UK.

Dor, D. and Zwick, U. (1999). SOKOBAN and Other Motion Planning Problems. *Computational Geometry: Theory and Applications*, Vol. 13, No. 4, pp. 215–228.

Doran, J. E. and Michie, D. (1966). Experiments with the Graph Traverser Program. *Proceedings of the Royal Society A*, Vol. 294, pp. 235–259.

Eppstein, D. (1987). On the NP-Completeness of Cryptarithms. *ACM SIGACT News*, Vol. 18, No. 3, pp. 38–40.

Eppstein, D. (2005). Nonrepetitive Paths and Cycles in Graphs with Application to Sudoku. Available from `http://arxiv.org/PS_cache/cs/pdf/0507/0507053.pdf`, last accessed 22 October 2006.

Fernau, H., Hagerup, T., Nishimura, N., Ragde, P., and Reinhardt, K. (2003). On the parameterized complexity of a generalized Rush Hour puzzle. *15th Canadian Conference on Computational Geometry*.

Flake, G. W. and Baum, E. B. (2002). Rush Hour is PSPACE-complete, or Why you should generously tip parking lot attendants. *Theoretical Computer Science*, Vol. 270, Nos. 1–2, pp. 895–911.

Fogel, D., Hays, T., Hahn, S., and Quon, J. (2006). The Blondie25 Chess Program Competes Against Fritz 8.0 and a Human Chess Master. *Proceedings of the Symposium on Computational Intelligence and Games*, pp. 230–235.

Frey, A. and Singmaster, D. (1997). *Handbook of Cubik Math*. Lutterworth Press. Reprinted in 2001.

Friedman, E. (2002a). Corral Puzzles are NP-complete. http://www.stetson.edu/˜efriedma/papers/corral.pdf. Last accessed 22 October 2006.

Friedman, E. (2002b). Pearl Puzzles are NP-complete. http://www.stetson.edu/˜efriedma/papers/pearl.pdf. Last accessed 22 October 2006.

Garey, M. and Johnson, D. (1979). *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Co., New York, NY.

Germundsson, R. (1991). A Tetris Controller – An Example of a Discrete Event Dynamic System. Available from citeseer.ist.psu.edu/germundsson91tetri.html, last accessed 22 October 2006.

Gupta, N. and Nau, D. (1992). On the Complexity of Blocks-World Planning. *Artificial Intelligence*, Vol. 56, Nos. 2–3, pp. 223–254.

Hallam, N., Poh, H. S., and Kendall, G. (2006). Using an Evolutionary Algorithm for the Tuning of a Chess Evaluation Function Based on a Dynamic Boundary Strategy. *In proceedings of 2006 IEEE International Conference on Cybernetics and Intelligent Systems*, pp. 1–6.

Hannsson, O., Mayer, A., and Yung, M. (1992). Criticizing Solutions to Related Models Yield Powerful Admissible Heuristics. *Information Sciences*, Vol. 63, No. 3, pp. 207–227.

Hearn, R. (2006). *Games, Puzzles, and Computation*. Ph.D. thesis, Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology.

Helmert, M. (2003). Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, Vol. 143, No. 2, pp. 219–262.

Herik, H. J. van den and Iida, H. (2000). *Games in AI Research*. Universiteit Maastricht, Maastricht, The Netherlands.

Hoogeboom, H. J. and Kosters, W. A. (2004). Tetris and Decidability. *Information Processing Letters*, Vol. 89, pp. 267–272.

Hoogeboom, H. J. and Kosters, W. A. (2005). The Theory of Tetris. *Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica (Newspaper of the Dutch Organization for Theoretical Computer Science)*, Vol. 9, pp. 14–21.

Jefferson, C., Miguel, A., Miguel, I., and Tarim, A. (2006). Modelling and Solving English Peg Solitaire. *Computers & Operations Research*, Vol. 33, No. 10, pp. 2935–2959.

Johnson, W. and Storey, W. (1879). Notes on the 15 Puzzle. *American Journal of Mathematics*, Vol. 2, No. 4, pp. 397–404.

Junghanns, A. (1999). *Pushing the Limits: New Developments in Single-Agent Search*. Ph.D. thesis, Department of Computing Science, University of Alberta.

Junghanns, A. and Schaeffer, J. (1998). Single-Agent Search in the Presence of Deadlocks. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 419–424.

Junghanns, A. and Schaeffer, J. (1999). Relevance Cuts: Localizing the Search. *Computers and Games: Proceedings CG'98*, Vol. 1558 of *Lecture Notes in Computer Science*, pp. 1–14.

Junghanns, A. and Schaeffer, J. (2001). Sokoban: improving the search with relevance cuts. *Theoretical Computer Science*, Vol. 252, Nos. 1–2, pp. 151–175.

Kalisker, T. and Camens, D. (2003). Solving Mastermind Using Genetic Algorithms. *Genetic and Evolutionary Computation Conference* (ed. E. Cant-Paz *et al.*), Vol. 2724 of *Lecture Notes in Computer Science*, pp. 1590–1591.

Kaye, R. (2000). Minesweeper is NP-complete. *The Mathematical Intelligencer*, Vol. 22, No. 2, pp. 9–15.

Kempe, D. (2003). On the Complexity of the Reflections Game. Available from http://www-rcf.usc.edu/~dkempe/publications/reflections.pdf, last accessed 22 October 2006.

Kendall, G. and Whitwell, G. (2001). An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics. *In proceedings of Congress on Evolutionary Computation*, pp. 995–1002.

Kendall, G. and Spoerer, K. (2004). Scripting the game of Lemmings with a Genetic Algorithm. *Proceedings of the Congress on Evolutionary Computation*, pp. 117–124.

Kiyomi, M. and Matsui, T. (2000). Integer Programming Based Algorithms for Peg Solitaire Problems. *Revised Papers from the Second International Conference on Computers and Games*, pp. 229–240. Springer-Verlag, London, UK.

Korf, R. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, Vol. 27, No. 1, pp. 97–109.

Korf, R. (1999). Sliding-tile puzzles and Rubik's Cube in AI Research. *IEEE Intelligent Systems*, pp. 8–14.

Korf, R. E. (1997). Finding optimal solutions to Rubik's Cube using pattern databases. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 700–705. Reprinted in *Games in AI Research*, H.J. van den Herik and H. Iida, Universiteit Maastricht, 1999, pp. 129-141.

Korf, R. E. and Felner, A. (2007). Recent Progress in Heuristic Search: A Case Study of the Four-Peg Towers of Hanoi Problem. *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 2324–2329.

Korf, R. E. and Schultze, P. (2005). Large-Scale Parallel Breadth-First Search. *In Proceedings of the 20th National Conference on Artificial Intelligence*, pp. 1380–1385.

Korf, R. and Felner, A. (2002). Disjoint pattern database heuristics. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 9–22.

Korf, R. and Taylor, L. (1996). Finding Optimal Solutions to the Twenty-Four Puzzle. *Proceedings of the National Conference on Artificial Intelligence*, pp. 1202–1207.

Kral, D., Majerech, V., Sgall, J., Tichy, T., and Woeginger, G. (2004). It is tough to be a plumber. *TCS: Theoretical Computer Science*, Vol. 313, No. 3, pp. 473–484.

Lewis, R. (2007). Metaheuristics can solve sudoku puzzles. *Journal of Heuristics*, Vol. 13, pp. 387–401.

Lloyd, S. (1959). *Mathematical Puzzles of Sam Lloyd*. Dover, New York, NY.

Lynas, A. and Stoddart, B. (2005). Sudoku Solver Case Study: from specification to RVM-Forth (part I). *Proceedings of the 21st EuroForth Conference*, pp. 21–34.

Matos, A. (1998). Depth-first search solves Peg Solitaire. Technical Report DCC-98-10, Universidade do Porto. Available from http://www.dcc.fc.up.pt/Pubs/treports.html, last accessed 22 October 2006.

McCarthy, J., Minsky, M., Rochester, N., and Shannon, C. (1955). Proposal for Dartmouth Summer Research Project on Artificial Intelligence. Technical report, Dartmouth College. Available at http://www-formal.stanford.edu/jmc/history/dartmouth.html, last accessed 22 October 2006.

McCarthy, J. (1998). Partial formalizations and the Lemmings game. Technical report, Stanford University, Formal Reasoning Group. http://www-formal.stanford.edu/jmc/lemmings.pdf, last accessed 22 October 2006.

McPhail, B. (2003). The Complexity of Puzzles: NP-Completeness Results for Nurikabe and Minesweeper. Thesis for BA, Division of Mathematics and Natural Sciences, Reed College.

McPhail, B. (2005). Light Up is NP-Complete. Available from http://www.reed.edu/~mcphailb/lightup.pdf, last accessed 22 October 2006.

Moore, C. and Eppstein, D. (2002). One-dimensional peg solitaire, and duotaire. *More Games of No Chance* (ed. R. Nowakowski), number 42, pp. 341–350. Cambridge University Press, New York, NY.

Morehead, A. A. and Mott-Smith, G. (2001). *The Complete Book of Solitaire and Patience Games*. Foulsham.

Murase, Y., Matsubara, H., and Hiraga, Y. (1996). Automatic Making of Sokoban Problems. *Pacific Rim International Conference on Artificial Intelligence*, pp. 592–600, Springer-Verlag, London, UK.

Neuwirth, E. (1982). Some strategies for mastermind. *Mathematical Methods of Operations Research*, Vol. 26, No. 1, pp. B257–B278.

Nicolau, M. and Ryan, C. (2006). Solving Sudoku with the GAuGE System. *Proceedings of the 9th European Conference on Genetic Programming*, pp. 213–224. Springer-Verlag, Berlin, Germany.

O'Rourke, J. and The Smith Problem Solving Group (1999). PushPush is NP-hard in 3D. Available from http://arxiv.org/abs/cs/9911013, last accessed 22 October 2006.

Ortiz-Garcia, E. G., Salcedo-Sanz, S., Perez-Bellido, A. M., and Portilla-Figueras, A. (2007). A Hybrid Hopfield Network-Genetic Algorithm Approach for the Lights-up Puzzle. *In proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1403–1407.

Papadimitriou, C., Raghavan, P., Sudan, M., and Tamaki, H. (1994). Motion planning on a graph (extended abstract). *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pp. 511–520.

Parberry, I. (1995). A real-time algorithm for the (n^2-1)-puzzle. *Information Processing Letters*, Vol. 56, No. 1, pp. 23–28.

Quartetti, C. (1998). Evolving a Program to Play the Game Minesweeper. *Genetic Algorithms and Genetic Programming at Stanford 1998*, pp. 137–146. Stanford University Bookstore, Stanford, CA.

Ratner, D. and Warmuth, M. (1990). Finding a Shortest Solution for the (N $\times$ N)-Extension of the 15-puzzle Is Intractable. *J. Symbolic Computation*, Vol. 10, pp. 111–137.

Ravikumar, B. (2004). Peg-Solitaire, String Rewriting Systems and Finite Automata. *Theoretical Computer Science*, Vol. 321, Nos. 2–3, pp. 383–394.

Reinefeld, A. (1993). Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA. *International Joint Conference on Artificial Intelligence*, pp. 248–253.

Rhee, S. (2000). Evolving Strategies for the Minesweeper Game using Genetic Programming. *Genetic Algorithms and Genetic Programming at Stanford 2000*, pp. 312–318.

Robertson, E. and Munro, I. (1978). NP-completeness, puzzles and games. *Utilas Mathematica*, Vol. 13, pp. 99–116.

Rodrigues Pereira, M., de Castro Dutra, I., and Clicia Stelling de Castro, M. (2001). Arc-consistency algorithms on a software dsm platform. *Colloquium on Implementation of Cronstraint and Logic Programming Systems - CICLOPS 2001*, pp. 117–131.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Eductation, second edition.

Sadikov, A. and Bratko, I. (2006). Learning Long-term Chess Strategies from Databases. *Machine Learning*, Vol. 63, No. 3, pp. 329–340.

Sastry, K., Goldberg, D., and Kendall, G. (2005). Genetic Algorithms. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (eds. E. Burke and G. Kendall), pp. 97–125, Springer-Verlag New York Inc.

Schadd, M. P. D., Winands, M. H. M., van den Herik, H. J., and Aldewereld, H. (2008). Addressing NP-Complete Puzzles with Monte-Carlo Methods. *AISB 2008 Convention: Communication, Interaction and Social Intelligence*.

Servais, F. (2005). Finding hard initial configurations of Rush Hour with binary decision diagrams. M.Sc. thesis, Université libre de Bruxelles, Faculté des sciences.

Siegel, E. and Chaffee, A. (1996). Genetically Optimizing the Speed of Programs Evolved to Play Tetris. *Advances in Genetic Programming 2*, pp. 279–298. MIT Press, Cambridge, MA, USA.

Singmaster, D. (1981). *Notes on Rubik's 'Magic Cube'*. Enslow Pub Inc.

Slaney, J. and Thiebaux, S. (2001). Blocks World Revisited. *Artificial Intelligence*, Vol. 125, pp. 119–153.

Slocum, J. and Sonneveld, D. (2006). *The 15 Puzzle Book*. Slocum Puzzle Foundation.

Spoerer, K. (2007). *The Lemmings Puzzle: Computational Complexity of an Approach and Identification of Difficult Instances*. Ph.D. thesis, University of Nottingham, School of Computer Science.

Stuckman, J. and Zhang, G.-Q. (2006). Mastermind is NP-Complete. *INFOCOMP Journal of Computer Science*, Vol. 5, pp. 25–28.

Takahiro, S. (2001). The complexities of puzzles, cross sum and their another solution problems (ASP). Thesis for BSc, Department of Information Science, University of Tokyo.

Taylor, L. and Korf, R. (1997). Pruning Duplicate Nodes in Depth-First Search. *National Conference on Artificial Intelligence*, pp. 756–761.

Uehara, R. and Iwata, S. (1990). Generalized Hi-Q is NP-complete. *Trans IEICE*, Vol. 73, pp. 270–273.

Walsh, R. and Julstrom, B. (1998). Generalized Instant Insanity: A GA-Difficult Problem. *Late Breaking Papers at the Genetic Programming 1998 Conference*, Stanford University Bookstore, Stanford, CA.

Yato, T. (2000). On the NP-Completeness of the Slither Link Puzzle. *IPSJ SIGNotes ALgorithms*, Vol. 74, pp. 25–32.

Yato, T. and Seta, T. (2003). Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 86, No. 5, pp. 1052–1060.

Yurovitsky, M. (1995). Playing Tetris Using Genetic Programming. *Genetic Algorithms and Genetic Programming at Stanford*, pp. 309–319. Stanford University Bookstore, Stanford, CA.