On the Complexity of the "Reflections" game

David Kempe

January 2, 2003

Abstract

The game "Reflections" is a puzzle game that has been around and entertaining users in various versions for many years. A single player tries to hit a given set of light bulbs with a laser beam, by placing mirrors and other optical items on grid cells. In this document, we investigate the complexity of the game, and show that deciding whether a given level has a solution is NP-complete. This is strong evidence that we are unlikely to come up with a simple way to solve levels of the game, suggesting that the game will provide more brain-teasing for its addicts. It puts the game in good company with other computationally hard puzzle games, including Minesweeper and Sokoban.

1 Introduction — The rules of Reflections

The game "Reflections" is a single-player puzzle game that has been around in various versions for many years, but recently experienced a surge in popularity due to its availability online. It is variously known by the name "Reflections"[8], "Aargon"[1] or "Laser!"[6]. These versions differ in the amount of gadgets and extensions they provide, but they share the same basic concept.

In this paper, we prove that deciding whether a given level of the game can be solved is NP-complete. (The reader unfamiliar with the concept of NP-completeness is referred to Appendix A.)

We present the rules as they are described for the version of the game called "Reflections"[8]. For the hardness results in Section 2, only very few gadgets are needed, so the results carry over easily.

In the game, the player puts *movable items* on a *board* B. The board is an $m \times n$ grid of square cells, with certain *fixed items* pre-installed. It contains exactly one source of light, also called the *laser*, whose beam must be reflected and split with the help of the movable items, so as to hit a set of *light bulbs* that are given as part of the board. Each of the bulbs must be hit an odd number of times, and in addition, none of the *bombs* that are part of the board must be hit by a beam.

In addition to the laser, light bulbs and bombs, the board may also contain *walls* and *one ways*. Walls do not let light pass through in any direction, but, other than the bombs, do not cause any damage if hit by light. The one ways let light traverse only in the direction that conforms with their direction, and act like walls otherwise. The orientation of all items in the game is always a multiple of 45°, i.e. the set of legal directions d is $D := \{\text{N, NW, W, SW, S, SE, E, NE}\}$. Since none of the items initially on the board can be rotated, the board can formally be regarded as a fixed mapping

$$B : \{1, \ldots, m\} \times \{1, \ldots, n\} \quad \rightarrow \quad \{\texttt{empty}, \texttt{bulb}, \texttt{bomb}, \texttt{wall}\} \cup (\{\texttt{laser}, \texttt{one-way}\} \times D).$$

Hence, the laser, too, is pointing in one of the eight legal directions. Notice that the square containing the laser is never entered by a light beam, only left by the source beam. Any ray does pass through light bulbs unaltered.

The player has at his disposition a multiset S of movable items, each of which is one of the following. A *mirror* has one reflective side, which reflects the light when hit at a 45° angle, and then reflects it physically correctly. If the mirror is hit at any other angle, it acts like a wall.[1] A *double mirror* has two reflective sides facing in opposite directions and behaving like a mirror each. The reflective sides are separated by a gap, so that if a double mirror is hit parallel to its faces, the ray passes through unaffected. A *splitter*, when entered

[1] Notice that this is in disagreement with the physical reality if the reflective side is hit frontally.

by its tip, splits a light ray into two rays leaving at right angles from the initial direction, to the left and right. If a splitter is entered at any other angle, it acts like a wall. Finally, the *refractor* can be rotated in two ways, such as to change the direction of the light beam by 45° to the left or right. If it is entered at any other angle, it too acts like a wall.

The player can rotate these items to any direction in D, and place them on any square of the board that was previously empty, i.e. contains no fixed items and no movable items. Throughout this process, the light source remains switched on, and the player looses if a light beam hits a bomb at any time. The player wins if each light bulb is hit an odd number of times by light beams, and no bomb was hit in the process of installing the movable items.

A *level* of the game consists of the board B and the set S of available items. We say that a level can be won if it is possible to install the movable items on the board as described above. In this paper, we show that deciding whether a level can be won is NP-complete.

2 Complexity of the game

This section is entirely devoted to proving the following theorem.

Theorem 1 *Given a level $\langle B, S\rangle$ of the reflections game, it is NP-complete to decide whether (a subset of) the items from S can be placed on grid points legally such that all lamps are hit by an odd number of light rays, and no bomb is hit by any light ray.*

Proof. Membership in NP

To show membership in NP, we want to show that the legality of a given solution can be verified in polynomial time. We define a graph G as follows: the vertices are pairs (z, d), where z is a square of the board, and d is one of the eight directions from D. We add directed edges between nodes of the graph. An edge $e = ((z_1, d_1), (z_2, d_2))$ is included if the cells z_1 and z_2 are adjacent (straight or diagonally), and, whenever light leaves the cell z_1 in direction d_1, then light also leaves cell z_2 in direction d_2. Notice that for the light source (z_0, d_0) (also called *source node*), by definition, light does not enter its cell from any direction (and hence cannot leave an adjacent cell in direction towards the source), so it has no incoming edges. The edges can be computed locally for each pair of adjacent cells, using the rules of the game.

Let G' be the graph consisting only of all nodes and edges reachable from (z_0, d_0) in G. G' can be computed in polynomial time using a simple breadth-first search. A crucial point is that the graph G' is in fact acyclic. Assume that it is not, and among all nodes that form part of a cycle, let (z, d) be the one with the smallest distance from the node (z_0, d_0). Because the source node has no incoming edges, this smallest distance is positive, and hence, the node (z, d) has an incoming edge belonging to the path from (z_0, d_0) to it. By minimality of the distance, this edge is distinct from the edge entering (z, d) that forms part of the cycle. That means that for two distinct directions of entering light, the light leaves in the same direction — which can be easily verified to be in conflict with the rules of the game.

Because G' is acyclic, a node is entered at most once through each edge, and because we removed all unreachable nodes and edges, also entered at least once through each edge. Now, we can compute, for each position z containing a bulb, the total number $\sum_d \delta^-(z, d)$ of light rays reaching that position, and verify that it is odd for all such z. In addition, we can verify that no node (z, d) is reachable such that in direction d of position z, a bomb is located (if there was, the assignment would not solve the level, either). These last computations, just like the breadth-first search, can be executed in polynomial time, proving membership in NP.[2]

The more interesting part is of course the NP-hardness of the problem, which we will be dealing with for the rest of the proof.

[2]Notice that when you actually play the game, the correctness of your solution is almost instantly verified, suggesting that the authors of the game are also aware of a polynomial-time verification algorithm.

The problem pn-planar 3SAT

We use a reduction from a special form of 3SAT that was proved NP-complete in [7]. Planar 3SAT is defined as follows: Let $\Phi = (X, C)$ be an instance of 3SAT, with variable set $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$ such that each clause consists of exactly 3 literals. Define a *formula graph* $G_\Phi = (V, E)$ with vertex set $V = X \cup C$, and edges $E = E_1 \cup E_2$, where $E_1 = \{(x_i, x_{i+1}) \mid i \leq n\}$, and $E_2 = \{(x_i, c_j) \mid c_j \text{ contains } x_i \text{ or } \overline{x}_i\}$.[3] That is, the graph G consists of all variables strung on a rosary, and connected to all clauses in which they occur.

A 3SAT formula Φ is called *planar* if the corresponding formula graph G_Φ is planar. One of the main results from [7] is that the set of planar 3SAT formulas is NP-complete. In Section 6 of his paper, Lichtenstein shows that the following more restricted version is still NP-complete. The edge set E_1 defines a cycle on the vertices X, and thus divides the plane into exactly two faces. Each node $c_j \in C$ lies in exactly one of those two faces.

We say that the formula is *positive-negative planar* (or, for short, *pn-planar*) if it is planar, and there exists a planar drawing of G_Φ such that if c_j and $c_{j'}$ contain opposite occurrences of the same variable x_i, then they lie in different faces of the plane. In other words, all clauses with literals x_i lie in one of the faces, and all clauses with $\overline{x}_i$ lie in the other face. By exchanging all occurrences of x_i and $\overline{x}_i$, we do not alter satisfiability, and obtain that all clauses on the inner face consist only of unnegated literals (in a slight abuse of terminology, we call them *unnegated clauses*), while all clauses on the outer face consist only of negated literals (*negated clauses*). This is the version of pn-planar 3SAT that we will be working with subsequently. We will make one additional assumption, namely that each variable x_i appears both negated and unnegated — we can ensure this by a simple preprocessing step that prunes away all variables that do not satisfy this assumption.

Obtaining a pn-planar grid drawing

Fix a pn-planar straight-line drawing of G, and assume w.l.o.g. that when visiting the x_i by order of increasing i, the unnegated clauses always lie on the left-hand side. Let ε be such that the distance between any two nodes in the drawing is at least ε, as is the distance between any two points that are part of non-adjacent edges (i.e. edges that do not share a vertex). We want to reduce the maximum degree of any vertex in the graph to 4, so that we can find a convenient grid embedding of the graph, i.e. one in which all nodes are points of the integer grid, and edges are paths in the grid. To achieve this, we will basically introduce a new vertex for each occurrence of a variable, and add a simple path through all unnegated occurrence vertices, as well as a simple path through all negated occurrence vertices.

Fix some variable x_i, and let α_i resp. β_i denote the number of unnegated resp. negated occurrences of x_i. Let $e_i^{(1)}, \ldots, e_i^{(\alpha_i)}$ be the edges connecting x_i to unnegated clauses, in *clockwise* order, and $f_i^{(1)}, \ldots, f_i^{(\beta_i)}$ the edges connecting x_i to negated clauses, in *counter-clockwise* order. If both $\alpha_i = 1$ and $\beta_i = 1$, then x_i has degree 4 already, and we do not need to do anything. So assume that $\alpha_i > 1$ (the changes for negated occurrences are similar and described below). For each $k \leq \alpha_i$, subdivide the edge $e_i^{(k)}$ with a new vertex $u_i^{(k)}$, at distance exactly $\varepsilon/2$ from the node x_i. Add edges $(u_i^{(k)}, u_i^{(k+1)})$ for $1 \leq k < \alpha_i - 1$, drawn as arcs on the circle of radius $\varepsilon/2$ around the node x_i (not straight lines). By the choice of the edge ordering (and ε), the resulting graph is still planar, and of course remains so after we remove the edges $(x_i, u_i^{(k)})$ for $2 \leq k \leq \alpha_i$. Finally, since node $u_i^{(\alpha_i)}$ has degree exactly two, it can be considered a mere subdivision point on the edge from $u_i^{(\alpha_i - 1)}$ to the clause node c_j incident with $u_i^{(\alpha_i)}$, and we "remove" the point, keeping only the edge $(u_1^{(\alpha_i - 1)}, c_j)$.

For the negative instances, if $\beta_i > 1$, we do exactly the same kind of construction as above, introducing new vertices $v_i^{(k)}$ in the process.

This construction ensures that the nodes $u_i^{(k)}$ and $v_i^{(k)}$ have degree exactly 3, and node x_i has degree exactly 4. The degrees of other nodes do not change. Notice that the resulting graph drawing has edges that are not straight lines, but all changes were made within distance $\varepsilon/2$ of node x_i, and hence will not

[3] Note that here and in the remainder, in order to improve legibility, we will assume that arithmetic "wraps around", i.e. that $n + 1 = 1$, and $1 - 1 = n$ etc.

interfere with changes made to other nodes $x_{i'}$. Let G' be the graph we obtain by applying this operation to all nodes x_i.

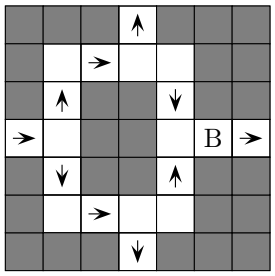
Because G' is planar and has maximum degree 4, we can use a result of Tamassia [9] that states that in polynomial time, we can obtain a grid embedding of G' with the minimum number of bends that preserves regions, i.e. that has a planar representation isomorphic to the one considered above. This representation is also known to fit in an $N \times N$ integer grid (where N is the number of vertices, in our case $N = O(nm)$). Notice that by the region preserving property, the representation is still pn-planar.

Constructing a level

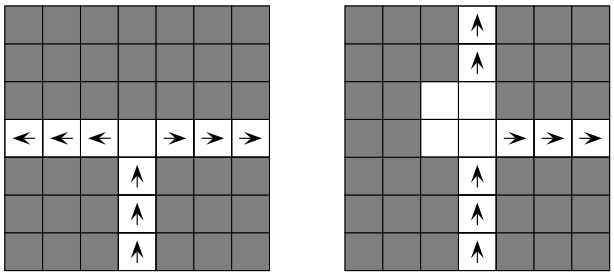
Now that we have the $N \times N$ grid graph drawing that we would like, we can go ahead and actually show how to turn it into a Reflections level. The crucial idea is to construct a variable gadget that forces a solution to choose between sending a light ray to all positive instances of the variable (thus effectively setting it to *true*), or to all negative instances (thus setting it to *false*), but does not allow a variable to be set to both *true* and *false*.

We "rescale" the grid by a factor of 7 along each axis. That is, each grid cell contains 7×7 game cells. Figure 1 depicts all the gadgets used in the reduction (except we use them in all four different rotations and mirrored form as necessary). They all have size 7×7 cells. The shaded cells denote walls, and the arrows are one-way cells. Light bulbs are denoted by the letter 'B'. We do not use any bombs in our hardness constructions.

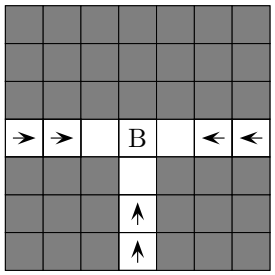
Later in the proof, we will be referring to individual cells of the gadgets with their coordinate — for instance, cell $(4, 7)$ of the variable gadget contains an arrow upward. These coordinates (and directions, such as top, right, etc.) will always be with respect to the drawing given in Figure 1, even though the actual instance of the gadget may be rotated or mirrored.



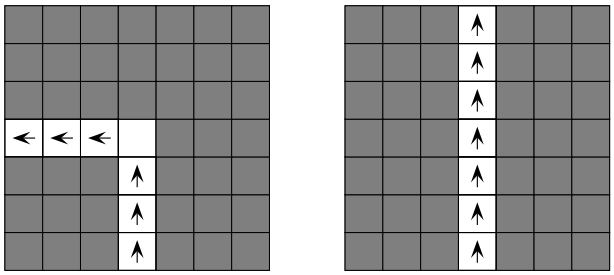
(a) The Variable Gadget



(b) The Instance Gadgets



(c) The Clause Gadget



(d) The Connection Gadgets

Figure 1: The gadgets used in the reduction

We take the pn-planar grid drawing, and construct from it the level $\langle B, S \rangle$. Whenever we speak of

replacing a grid point by a gadget, we mean that we add a copy of the gadget to the grid such that its center lies on the grid point.

Each node x_i is replaced by a variable gadget, which is rotated such that the one incoming arrow lines up with the last segment of the edge (x_{i-1}, x_i). Notice that this will automatically line up the right outgoing arrow with the first segment of the edge (x_i, x_{i+1}).

The nodes $u_i^{(k)}$ for $1 \leq k < \alpha_i$ and $v_i^{(k)}$ for $1 \leq k < \beta_i$ are replaced with instance gadgets. They are rotated and mirrored such that the incoming arrow lines up with the incident segment of the edge $(u_i^{(k-1)}, u_i^{(k)})$ for $k > 1$ (or with the incident segment of the edge $(x_i, u_i^{(1)})$ for $k = 1$), and the outgoing arrows line up with the other two incident edge segments. (For negative instances, the incoming arrow lines up with the edge segments of $(v_i^{(k-1)}, v_i^{(k)})$ resp. $(x_i, v_i^{(1)})$.)

Each clause node c_j is replaced by a copy of the clause gadget, rotated so that the three incoming arrows match up with the incident edge segments.

Each grid point with incident edges that was not a node in our graph (i.e. grid points that lie on edges) is replaced by one of the two connection gadgets, rotated and mirrored so that it lines up with the edge segments incident with that point. Edges between variable nodes are directed from x_i to x_{i+1}. Edges between instance nodes are directed from $u_i^{(k)}$ to $u_i^{(k+1)}$ (or from $v_i^{(k)}$ to $v_i^{(k+1)}$). Edges between variable and instance nodes are directed towards the instance node, and edges between instance and clause nodes are directed towards the clause node.

Finally, the incoming arrow of the first variable node x_1 is replaced by the light source, pointing in the same direction as the arrow. This describes the construction of the board. To complete the reduction, we must specify the set S of available items. For these, we use no refractors, no double mirrors, and an essentially "unlimited" supply of single mirrors and splitters, such as $(7N)^2$ of each of them (this is the maximum number that could be put on the board in either case).

Let us quickly verify that the construction can be performed in polynomial time. The straight line drawing can be obtained in polynomial time (e.g. using an algorithm by Hopcroft and Tarjan [4]). The graph modifications are obviously done in polynomial time, and by a result of Tamassia [9], a region preserving grid drawing can be found in polynomial time, and its grid size is polynomial in m, n. The final graph alterations and level constructions also take polynomial time, so the entire construction is polynomial. Now on to the more interesting stuff — verifying that it is actually a correct reduction.

Satisfiable formulas

We want to show that if Φ has a satisfying assignment, then the corresponding level $\langle B, S\rangle$ can be solved. Fix a satisfying assignment $(y_i), i = 1, \ldots, n$ to variables. We will simply exhibit a solution to the level. For the variable gadget corresponding to variable x_i, we add the following movable items:

- A splitter at position $(2, 4)$, with the tip facing to the left.
- Mirrors at positions $(2, 2), (2, 6), (5, 2), (5, 6)$, with the reflective side facing NE, SE, NW and SW, respectively.
- If y_i is *true*, a mirror facing NW at position $(4, 6)$, and a mirror facing SE at position $(5, 4)$.
- If y_i is *false*, a mirror facing SW at position $(4, 2)$, and a mirror facing NE at position $(5, 4)$.

It is easy to verify that this assignment ensures that there always is exactly one light ray hitting the bulb, and leaving the gadget to the right. In addition, if y_i is *true*, then there is a light ray leaving the gadget at the top, and otherwise, there is a light ray leaving at the bottom.

For straight connection gadgets, we need not (and for that matter, cannot) add any items. For the angled connection gadget, we add a mirror facing NW at position $(4, 4)$. Then, as soon as a ray of light enters a connection gadget, it will leave at the exit. Because light leaves every variable gadget at the right exit, the connection gadgets ensure that each variable gadget is indeed entered, as is the first instance gadget of every literal set to *true*.

For the left version of the instance gadget, we simply place a splitter at the one open position, pointing downwards. For the right version, we place a splitter at position $(4, 4)$, and mirrors at positions

$(3,4),(3,5),(4,5)$, facing NE, SE and NW, respectively. This ensures that once a light ray enters the gadget, a light ray leaves through both exits. Since we ensured that the first instance gadget was entered, all instance gadgets are entered inductively. Also, there is a ray leaving each exit of each instance gadget corresponding to a true literal, and hence entering its corresponding clause gadget.

Because the variable assignment was assumed to satisfy the formula, there is now at least one ray entering each clause gadget. By placing mirrors (with any rotation) at one or two of the positions $(3,4),(4,3),(5,4)$, we can easily ensure that exactly one of those rays hits the light bulb. Altogether, this will ensure that each bulb is hit exactly once, and because there were no bombs in the level to begin with, this solves the level.

Unsatisfiable formulas

Now, we prove the converse direction. Assume that there is a way of placing items on the board to solve the level. Then, all light bulbs are hit an odd number of times, in particular, they are all hit at least once. Using this solution, we define a variable assignment $(y_i), i = 1,\ldots,n$. Set y_i to *true* if there is a ray of light leaving the gadget for x_i at the top, and to *false* if there is a ray of light leaving it at the bottom. If no ray of light is leaving at either the top or the bottom, set y_i arbitrarily.

The crucial part is to verify that this is a well-defined assignment, i.e. that a variable is not simultaneously set to *true* and *false*. Assume that it were, i.e. there are rays leaving the gadget both at the top and the bottom. For the bottom ray, consider the square $(4,2)$. Because that square can only be entered from the left, it must contain either a mirror or a splitter, changing the direction of the ray at a right angle. For either of them, no ray will enter the square $(5,2)$. A similar argument for the top ray shows that no ray enters the square $(5,6)$. But these two are the only squares via which the square $(5,4)$, and with it the bulb, can be entered, so the bulb could not have been hit by a ray, a contradiction. Hence, the assignment is well-defined.

Now, we want to show that it actually satisfies the formula Assume it does not, and let $c = x_1 \vee x_2 \vee x_3$ be a clause that is not satisfied (due to symmetry, we may assume w.l.o.g. that the clause is unnegated — the choice of variable names is only for notational convenience). Because the bulb in the gadget for c is hit at least once, the gadget must be entered by a ray of light. Tracing back the connection gadgets, we see that this implies that for at least one literal from the clause, an instance gadget for that literal must be entered by a ray of light. Assume that $u_1^{(i)}$ is that gadget.

Let k be minimal such that the gadget for $u_1^{(k)}$ is entered by light (k is defined, because i is a candidate). If $k > 1$, then by minimality, the gadget for $u_1^{(k-1)}$ was not entered. But the only way for light to enter the gadget for $u_1^{(k)}$ is by exiting the one for $u_1^{(k-1)}$, a contradiction. For the gadget $u_1^{(1)}$, the only way for light to enter it is by exiting the gadget for x_1 at the top. But light cannot have exited that gadget at the top, because c was assumed to not be satisfied, and in particular, x_1 must have been set to *false* by the assignment. In either case, we obtain a contradiction, and hence, the assignment is satisfying, completing the proof. ■

3 Conclusions

We have proved that the popular puzzle game Reflections is NP-complete to solve in general. That means that, unless P = NP, we do not expect to see any solvers for the game that can solve all levels efficiently, in polynomial time. In particular, no simple rule for solving all levels is to be expected, hence guaranteeing some more brain-teasing fun for human players.

This puts the game in good company. Recently, Kaye[5] showed that deciding whether a safe move exists for a given Minesweeper-configuration is also NP-complete. Culbertson[2] showed that Sokoban is PSPACE-complete, making it even harder than Minesweeper or Reflections. It is well-known that solving partially completed Latin Squares is NP-complete, and it can be easily shown that the same applies to Jigsaw Puzzles (see Appendix B).

Empirical observations lead us to believe that for a puzzle game to be entertaining in the long run, NP-hardness is a desirable property, since it means that likely, no simple (or boring) strategy exists for solving the game. We can expect a lot more entertainment from the Reflections game.

Acknowledgements

I would like to thank Lori Lorigo for pointing me to the game "Reflections".

References

[1] `http://www.twilightgames.com`

[2] J. Culbertson. "Sokoban is PSPACE-complete," *Univ. of Alberta Comp. Sci. Dept. Technical Report* TR97-02.

[3] M. Garey, D. Johnson. "Computers and Intractability. A guide to the Theory of NP-Completeness," *Freeman* 1979.

[4] J. Hopcroft, R. Tarjan. "Efficient planarity testing," *J. of the ACM* 21 (1974).

[5] R. Kaye. "Minesweeper is NP-complete," *Mathematical Intelligencer* 22 (2000).

[6] `http://www.microserve.net/~ronb/`

[7] D. Lichtenstein. "Planar formulae and their uses," *SIAM J. on Computing* 11 (1981).

[8] `http://www.input-entertainment.de/laser`

[9] R. Tamassia. "On embedding a graph in the grid with the minimum number of bends," *SIAM J. on Computing* 16 (1987).

A Reductions and NP-completeness

This appendix provides a brief and slightly informal introduction to the concepts of polynomial-time reductions and NP-completeness. To the reader interested in more detail, we recommend the wonderful book by Garey and Johnson[3].

In theoretical computer science, tractability of a problem is usually equated with being solvable in polynomial time. That is, we should have an algorithm that solves all instances of the problem in time $c \cdot n^k$ for some constants c and k that may be arbitrarily large, but not depend on n. For instance, n could be the number of symbols that we need to write down a description of a level of the Reflections game. We would then like to be able to compute a solution in time $c \cdot n^k$. The results of this paper suggest that we are unlikely to find an algorithm doing that. The class of all problems that can be solved in polynomial time is denoted by P.

As puzzle solvers know, coming up with a solution is often quite difficult, whereas understanding or verifying a solution after seeing it usually seems considerably easier. The notion of being "easy to verify" is captured formally by the class NP of *non-deterministic polynomial time*. A problem is in this class if it is possible, in polynomial time, to verify that a proposed solution to the problem is correct. For instance, while it is hard to decide whether a level of the Reflections game has a solution, it is much easier to decide whether a solution that someone showed us is correct. Part of the result from Section 2 is that indeed, the correctness of a solution can be verified in polynomial time.

One of the biggest open problems in theoretical computer science and mathematics to this day is whether these two classes are equal, i.e. whether P = NP. Although literally thousands of scientists have been working on the problem over the past thirty years, it seems like we are still far from resolving that question. Obviously, verifying a solution is no more difficult than coming up with it, so P $\subseteq$ NP, but the other direction is unknown.

Throughout the history of mathematics, it has always been an important problem-solving technique to reduce one problem to another. That is, we take a problem A, apply some transformations, and obtain a new problem B, from the solution of which we can reconstruct the solution to the initial problem A. For one thing, this might help us to solve A. For another, if we didn't put a lot of work into the transformation, it shows that B is no easier than A, since we can use it to solve A.

For problems in NP, we make the term "a lot of work" more specific by restricting reductions to be computed in polynomial time. That is, we are given a problem A, that is written down in n symbols, and apply a transformation that has to finish in $c \cdot n^k$ steps, to obtain a new problem B. Let us call the transformation function f. Then, if problem B were in P, then so would problem A, because to solve it, we could apply f, and then solve B. That is, by exhibiting a polynomial-time reduction f from A to B, we can show that problem B is at least as hard as A.

A problem B is called NP-hard, if it is at least as hard as all of the problems in NP, i.e. there is a polynomial-time reduction from A to B for all $A \in$ NP. It is called NP-complete if it is NP-hard, and also contained in NP itself. In a sense, the NP-complete problems are the hardest problems in NP. The interesting fact is that not only do NP-complete problems exist, but there are many such problems, and they are very natural and come up in real-world situations. They include the following:

3SAT: Given a formula Φ of the form $\Phi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, where each c_i is of the form $l_{i,1} \vee l_{i,2} \vee l_{i,3}$ with literals $l_{i,j}$ that are either x_j or $\overline{x}_j$ for some variable x_j, can we assign the values *true* and *false* to the variables such that the formula is true?

Traveling Salesman: Given a map with cities, and a maximum distance d, is there a route of total length at most d that visits all cities?

Partition: Given a set of coins with their values, can they be partitioned into two heaps of equal total value?

For all these, and many more problems, no algorithm running in less than exponential time is known, despite years of research efforts. However, neither is there any proof that it is not possible to devise polynomial-time algorithms for these problems. The fact that all of these problems are NP-complete means that if we had a polynomial-time algorithm for one of them, we would have algorithms for all of them (via the reductions). On the other hand, if we could prove that one of them cannot be solved in polynomial time, we would have a proof that none of them can.

For the reader wishing to see an easier reduction and NP-completeness proof before delving into the one for the Reflections game, we recommend reading the NP-completeness proof for Jigsaw Puzzles in the next section.

B NP-completeness of Jigsaw Puzzles

In a Jigsaw puzzle, we are given pieces in various shapes, and have to place them on a surface so as to form a desired image. For the sake of proving the hardness, we will abstract away the image, and pose the problem as follows: Given a set of polygonal shapes $s_1, \ldots, s_n$, can they be placed so as to form a $w \times h$ rectangle? We will prove the following fact about Jigsaw Puzzles:

Fact 2 *It is NP-hard to decide whether a given Jigsaw Puzzle can be solved.*

Proof. To show that the problem is NP-hard, we have to show that it is at least as hard as some other known NP-hard problem. We choose the Partition problem.

An instance of the partition problem consists of n natural numbers $a_1, \ldots, a_n$, and we have to decide whether there is a subset S of $\{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = \sum_{j \notin S} a_j$.

For the reduction, we generate n rectangles, where the ith rectangle has size $1 \times a_i$. We then set $w = 2$ and $h = \frac{1}{2} \cdot \sum_{i=1}^{n} a_i$. This obviously only takes polynomial time.

Now assume that there is a solution S to the Partition instance. Then, we put the rectangles corresponding to $i \in S$ in the left column, and the other ones in the right column. Because S is a solution, both columns have equal height, namely h.

On the other hand, if there is a solution to the Jigsaw puzzle, then it must form exactly two columns of width 1 each. We can define S to be the set of all i such that the rectangle obtained from a_i is in the left column. Because both columns must have height h, the sums of the a_i values for $i \in S$ and $i \notin S$ are equal.

This proves that the reduction was indeed correct, and since it runs in polynomial time, we showed that the Jigsaw problem is NP-hard. ∎

In fact, as formulated, the Jigsaw problem is also in NP, but this is not our concern here.