The Fewest Clues Problem

Erik D. Demaine* Fermi Ma† Ariel Schvartzman†
Erik Waingarten‡ Scott Aaronson§

February 18, 2019

Abstract

When analyzing the computational complexity of well-known puzzles, most papers consider the algorithmic challenge of *solving* a given instance of (a generalized form of) the puzzle. We take a different approach by analyzing the computational complexity of designing a "good" puzzle. We assume a puzzle maker designs part of an instance, but before publishing it, wants to ensure that the puzzle has a unique solution. Given a puzzle, we introduce the FCP (fewest clues problem) version of the problem:

> Given an instance to a puzzle, what is the minimum number of clues we must add in order to make the instance uniquely solvable?

We analyze this question for the Nikoli puzzles Sudoku, Shakashaka, and Akari. Solving these puzzles is NP-complete, and we show their FCP versions are Σ_2^P-complete. Along the way, we show that the FCP versions of Triangle Partition, Planar 1-in-3 SAT, and Latin Square are all Σ_2^P-complete. We show that even problems in P have difficult FCP versions, sometimes even Σ_2^P-complete, though *"closed under cluing"* problems are in the (presumably) smaller class NP; for example, FCP 2SAT is NP-complete.

Keywords: computational complexity, pencil-and-paper puzzles, hardness reductions

1 Introduction

A natural aesthetic for designing pencil-and-paper puzzles is to provide as few starting hints as possible, yet have the resulting solution be unique. For example, in the well-known Sudoku puzzle, the starting configuration is a partially filled 9×9 grid where the goal is to fill in the remaining entries. Hard Sudoku puzzles tend to give very few numbers in the starting configuration, but must still give enough numbers to ensure that the puzzle has a unique solution. The natural question here is, how small can we make the starting configuration?

In this paper, we formalize this question by the family of "Fewest Clues Problem" (FCP). Given an NP search problem where the certificate is written as a string, we ask: is there a setting of at most k characters of the certificate such that there exists exactly one way to complete the certificate (fill in the remaining characters)?

FCP seems superficially related to two other classes of problems, the "Another Solution Problem" (ASP) and counting (#) problems [8, 11, 12, 13, 14]. The ASP problem asks:

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St, Cambridge, MA 02139, USA, edemaine@mit.edu

†Department of Computer Science, Princeton University, 35 Olden St, Princeton, NJ 08544, USA, fermim@princeton.edu,acohenca@princeton.edu

‡Department of Computer Science, Columbia University, 1214 Amsterdam Ave, New York, NY 10027, USA, eaw@cs.columbia.edu

§Department of Computer Science, University of Texas at Austin, 2317 Speedway, Austin, TX 78712 USA, aaronson@cs.utexas.edu

given an instance as well as one solution to a search problem, is there another solution? Counting problems ask: given an instance, how many solutions are there? All three of these problem types are transformations of an original NP search problem.

ASP versions of NP-hard problems are in NP by definition, and interestingly, there are NP-hard problems whose ASP versions are NP-hard as well. The counting version of a problem can be significantly harder than the original problem. It is well known that some problems in P (such as 2SAT) have #P-hard counting versions, and Toda's theorem states that such problems are as hard as the polynomial hierarchy [10].

Our results. This paper has three main contributions:

1. We introduce the FCP framework for analyzing puzzles, and we develop simple techniques to adapt existing hardness reductions to the FCP setting (Section 2). With these techniques, we show that the FCP versions of common NP-complete problems are Σ_2^P-complete. These include FCP Planar 1-in-3 SAT, FCP Triangle Partition, FCP Planar 3SAT, and FCP Latin Square (Section 3.2).

2. We show that the FCP versions of three common Nikoli puzzles (Sudoku, Shakashaka, and Akari) are Σ_2^P-complete by reducing from the above problems (Section 4). Figure 1 shows a chain of reductions which allow us to conclude these problems are Σ_2^P-complete.

3. We analyze how the computational complexity of problems in P changes when we consider their FCP versions (Section 5). For a class of problems *closed under cluing*, their FCP versions are in NP. In addition, we show FCP 2SAT, which naturally falls in the class of problems closed under cluing, is NP-complete. We give an example of a problem in P whose FCP version is Σ_2^P-complete.

2 Definitions and Overview

2.1 FCP Framework

Definition 1 *For each $A \in$ NP, we associate an NP* relation*, R_A the set of instance-certificate pairs.*

For example, the NP relation for the Boolean satisfiability problem (SAT) consists of pairs (ϕ, x) where ϕ is a Boolean formula and x is an assignment to the variables where $\phi(x)$ evaluates to true. We assume that instances and certificates are strings from some alphabet Σ. From now on, when referring to some $A \in$ NP, we implicitly consider the NP relation, R_A.

If the string x is a solution to a SAT problem, we call a "partially filled" version of x (constructed by blocking out certain characters of x with a $\perp$ symbol) a *clue* for that problem. We define this notion for all NP problems as follows.

Definition 2 *For any instance I of a problem $A \in$ NP, a string $c = (c_i) \in (\Sigma \cup \{\perp\})^*$ is a* clue *if there exists some certificate string $y = (y_i) \in \Sigma^*$ with $(I, y) \in A$ such that if $c_i \neq \perp$, then $c_i = y_i$. The size of a clue is the number of non-$\perp$ characters. We refer to y as a solution which* satisfies *the clue c, and denote this as $c \subset y$.*

Definition 3 *For $A \in$ NP, FCP A is the following decision problem:*

> Given $x \in \Sigma^*$ and an integer k, does there exist a clue c of size at most k with only one satisfying solution to x?

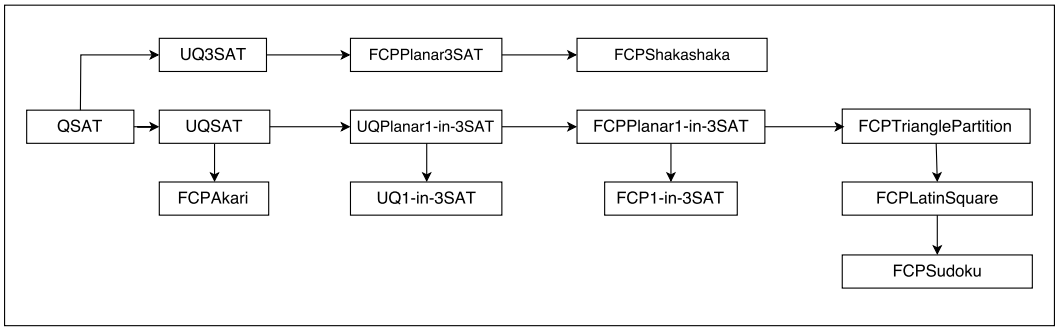


Figure 1: Chain of reductions from QSAT to FCP versions of satisfiability problems and FCP versions of pencil-and-paper puzzles.

2.2 FCP Versions of Classic Problems

In Section 3.1, we show that the FCP version of any NP problem is in Σ_2^P. Our more compelling result is that the FCP versions of many NP-complete problems are themselves Σ_2^P-complete. We show this by using a variety of techniques that modify existing NP-completeness reductions to give Σ_2^P-completeness results for the corresponding FCP problems.

We first prove Σ_2^P-completeness of the FCP versions of two common SAT variants. This will be useful since many known NP-completeness reductions are from variants of SAT. We show that FCP Planar 3SAT and FCP Planar 1-in-3 SAT are both Σ_2^P-complete. We obtain these hardness results with a chain of reductions, starting with the canonical Σ_2^P-complete problem, quantified satisfiability with one pair of alternating quantifiers, $\exists$ and $\forall$ (QSAT). The chain of reductions for all problems defined in this subsection and Section 2.3 are shown in Figure 1.

A slight issue with constructing this chain of reductions is that QSAT does not have any notion of uniqueness, which is essential in FCP problems. We overcome this by defining the unique quantified satisfiability problem (UQSAT), a problem we show is Σ_2^P-complete. Unfortunately, the reduction will introduce clauses which are longer than three literals. For our hardness reductions, we often want to work with 3-CNF formulae. Thus we introduce UQ3SAT, another provably Σ_2^P-complete problem.

For the following problems, $\phi(x, y)$ is a Boolean function in CNF form on the sets of variables x and y.

> QSAT: Given $\phi(x, y)$, does there exist an assignment of the variables in x, such that for all assignments of variables in y, $\phi(x, y) = 1$?
>
> UQSAT: Given $\phi(x, y)$, does there exist an assignment of the variables in x, such that there exists a unique assignment of the variables in y with $\phi(x, y) = 1$?
>
> UQ3SAT: Given $\phi(x, y)$ in 3-CNF form, does there exist an assignment of the variables in x, such that there exists a unique assignment of the variables in y with $\phi(x, y) = 1$?
>
> UQ Planar 3SAT: Given a planar $\phi(x, y)$ in 3-CNF form, does there exist an assignment of the variables in x, such that there exists a unique assignment of the variables in y with $\phi(x, y) = 1$?

UQ Planar 1-in-3 SAT: Given a planar $\phi(x, y)$ in 3-CNF form, does there exist an assignment of the variables in x, such that there exists a unique assignment of the variables in y with $\phi(x, y) = 1$, where each clause has exactly one true literal?

The transition to FCP problems relies on the following observation: while UQ3SAT specifies which variables can be assigned, FCP 3SAT allows any k variables to be assigned.

We address this distinction by building *assign gadgets*; these gadgets are built so that specifying some part of them fixes in place other aspects of the reduction. These assign gadgets also have the property of having inherent ambiguity. Thus, any clue must specify something within the assign gadget to resolve this ambiguity, or else a unique solution will not be possible. If we place exactly k assign gadgets, we will need a clue of size at least k. This technique will be useful for reducing from non-FCP problems to FCP problems.

For the following problems, $\phi(x)$ is a Boolean formula in 3-CNF form.

FCP 1-in-3 SAT: Given $\phi(x)$ and a number k, does there exist a partial assignment of at most k variables such that the remaining formula ϕ' has only one satisfying assignment, where each clause has exactly one true literal?

FCP Planar 3SAT: Given a planar $\phi(x)$ and a number k, does there exist a partial assignment of at most k variables such that the remaining formula ϕ' has only one satisfying assignment?

FCP Planar 1-in-3 SAT: Given a planar $\phi(x)$ and a number k, does there exist a partial assignment of at most k variables such that the remaining formula ϕ' has only one satisfying assignment, where each clause has exactly one true literal?

We use assign gadgets to show that FCP Planar 3SAT and FCP Planar 1-in-3 SAT (and hence FCP 1-in-3 SAT) are Σ_2^P-complete.

In some cases, an NP-hardness reduction from problem A to problem B may already preserve clue structure without needing any modifications. If FCP A is Σ_2^P-hard, then the same reduction implies that FCP B is Σ_2^P-hard.

2.3 FCP Versions of NP-hard Puzzles

In this section, we introduce a number of NP-hard pencil-and-paper puzzles. We modify NP-hardness reductions for these problems to show that their FCP versions are Σ_2^P-hard. These problems were chosen because their NP-hardness reductions mostly preserve clue structure. Figure 1 summarizes the chain of reductions.

FCP Latin Square: Given a partially filled Latin Square and a number k, do there exist k additional squares to fill in such that the remaining puzzle has a unique solution?

FCP Sudoku: Given a partially filled Sudoku board and a number k, do there exist k additional squares to fill in such that the remaining puzzle has a unique solution?

FCP Shakashaka: Given a Shakashaka board and a number k, do there exist k clues for the board such that the remaining board has a unique solution?

FCP Akari: Given an Akari board and a number k, do there exist k clues to the board such that the remaining board has a unique solution?

There is a fundamental difference between two groups of problems: (1) Shakashaka and Akari, and (2) Sudoku and Latin Squares. In the former, there is a clear distinction between problem instance and solution, whereas in the latter, this distinction is less clear. In particular, filling in a clue in Sudoku and Latin Squares simply produces a new instance of the problem, whereas filing in clues for Shakashaka and Akari do not create new problem instances.

2.4 FCP Versions of Easy Problems

It is also interesting to consider the FCP transformation applied to problems in P. Here, we do not find many general hardness bounds; instead, the results are problem-dependent.

We provide an upper bound for the class of problems in P where filling in a clue results in another instance of the problem. The FCP versions of such problems are always in NP. As an example of such a problem, consider 2SAT.

> FCP 2SAT: Given a Boolean formula $\phi(x)$ written in 2-CNF form, and a number k, does there exist a partial assignment of at most k variables such that the remaining formula ϕ' has only one satisfying assignment?

In this problem, filling in a clue means assigning truth values to certain variables of ϕ, which results in another instance of 2SAT. Thus, our upper bound will imply that FCP 2SAT is in NP. In fact, this bound is tight as we show that FCP 2SAT is NP-complete by reduction from Minimum Independent Dominating Set.

However, for problems in P that do not have this property, the hardness of their FCP versions cannot be so easily characterized. This happens because we can modify hard problems by planting solutions to make them "easy", while maintaining the hardness of their FCP versions. We give a simple example of a problem in P whose FCP version is Σ_2^P-hard.

3 The FCP Transformation

3.1 Upper Bounds

Theorem 4 *If $L \in$ NP, then FCP $L \in \Sigma_2^P$.*

Proof: We write FCP L as an instance of QSAT, a problem in Σ_2^P. The original problem of deciding membership in L can be written as

$$x \in L \leftrightarrow \exists y : A(x, y) = 1,$$

where A is a polynomial time algorithm and $|y|$ is polynomial in $|x|$.

FCP L asks: for a given instance x and positive integer k, does there exist a clue c of size at most k such that there is only one solution y with $c \subset y$ and $(x, y) \in R$? We write this in logical notation as

$$(x, k) \in \text{FCP } L \leftrightarrow \exists c, y : \forall y' : A'(x, c, y, y') = 1.$$

A' simply checks that $c \subset y$, c is of size at most k (c is a valid clue of the correct size), $c \subset y' \neq y$ (y' is another possible solution), $A(x, y) = 1$ (y is a solution), and $A(x, y') = 0$ (all other possible solutions do not work). □

3.2 Lower Bounds

In this subsection we present a chain of Σ_2^P-hardness reductions starting with UQSAT and UQ3SAT. These results allow us to transition to Σ_2^P-hardness proofs of various FCP problems.

Lemma 5 UQSAT *is* Σ_2^P*-hard.*

Proof: We reduce from QSAT. We define a function f from instances of QSAT to instances of UQSAT which maps $\phi(x, y)$ to $\phi'(x, y, z)$, where z is one additional variable. We show

$$\exists x : \forall y : \phi(x, y) = 1 \iff \exists x : \phi'(x, y, z) \text{ has a unique solution.}$$

Let $\phi'(x, y, z) = (\overline{z} \wedge \bigwedge_i \overline{y_i}) \vee \overline{\phi(x, y)}$.

Note that for each assignment of x, $\phi'(x, 0, 0) = 1$. If there exists some x for which $\phi(x, y) = 1$ always, then for this x, $\overline{\phi(x, y)} = 0$ always. For this x, the only way to satisfy ϕ' is to set z and each y_i to 0. It remains to rewrite ϕ' as a CNF formula.

The given $\phi(x, y)$ is in CNF form, so we write $\overline{\phi(x, y)}$ in DNF form using De Morgan's laws. For the ith clause, we introduce the variable c_i to represent whether that clause is satisfied. If there are n clauses, so the original instance can be written as $\phi(x, y) = (\overline{c_1} \wedge \overline{c_2} \wedge \cdots \wedge \overline{c_n})\overline{\phi(x, y)}$ becomes $(c_1 \vee c_2 \vee \cdots \vee c_n)$. We write the jth literal in the ith clause as $l_{i,j}$, and introduce A as one additional variable. Then the following formula is logically equivalent to ϕ':

$$\phi''(x, y, z, \{c_i\}, A) = (\overline{A} \vee \overline{z}) \wedge \bigwedge_i (\overline{A} \vee \overline{y_i}) \wedge \bigwedge_i \bigwedge_j (\overline{c_i} \vee l_{i,j}) \wedge (A \vee \bigvee_i c_i).$$

Then the following two claims complete the proof.

Claim. Suppose for some assignment x, $\phi(x, y) = 1$ for all y. Then $\phi''(x, y, z, \{c_i\}, A)$ has a unique satisfying assignment.

Since $\phi(x, y) = 1$, $\overline{\phi(x, y)} = 0$ and $c_i = 0$ for all i. This further implies that each $l_{i,j} = 0$. This forces $A = 1$, $z = 0$, and $y_i = 0$ for all i.

Claim. If for each x there exists some y where $\phi(x, y) = 0$, then after fixing x, $\phi''(x, y, z, \{c_i\}, A)$ has more than one satisfying assignment.

Setting each $y_i = 0$, $z = 0$, $c_i = 0$ for each i and $A = 1$ gives one satisfying assignment. If there is a setting of the y_i's which makes some of the clauses true, then for the true clauses set $c_i = 1$. Then set $A = 0$ and z to either 0 or 1. This gives additional satisfying assignments for ϕ''.

□

Lemma 6 UQ3SAT *is* Σ_2^P*-hard.*

Proof: The reduction is from UQSAT. Given $\phi(x, y)$, an instance of UQSAT, it remains to transform $\phi(x, y)$ into 3-CNF form. For clauses in $\phi(x, y)$ of length $m > 3$, we imagine a binary tree of height $\log m$ where leaves correspond to clause literals. For all interior nodes of the tree, we introduce a new variable to act as the OR of its two children. If an interior node has children i_1 and i_2, we introduce the variable o along with the following clauses:

$$(\overline{i_1} \vee \overline{i_2} \vee o), \quad (\overline{i_1} \vee i_2 \vee o), \quad (i_1 \vee \overline{i_2} \vee o), \quad (i_1 \vee i_2 \vee \overline{o}).$$

These clauses ensure that o is the OR of i_1 and i_2. Additionally, we add the variable at the root of the tree as a new clause of size 1 to capture the original clause exactly.

To handle clauses with fewer than three literals, we pad the clause with the additional variable a along with the clause $(\overline{a} \vee \overline{a} \vee \overline{a})$ to ensure that a is set to false. □

Lemma 7 UQ Planar 3SAT *is* Σ_2^P*-hard.*

Proof: The reduction is from UQ3SAT. We can apply the crossover gadget in [5] to obtain an instance of UQ Planar 3SAT. Since the gadget is parsimonious, the uniqueness of the solutions is preserved. □

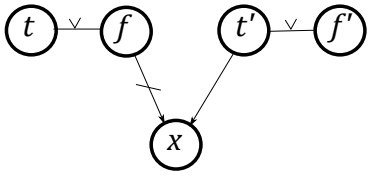


Figure 2: FCP PLANAR 3SAT assign gadget for variable x. The above diagram is written with 4 clauses: $(t \vee f) \wedge (\overline{f} \vee \overline{x}) \wedge (\overline{t'} \vee x) \wedge (t' \vee f')$. The arrows indicate "causal" relationships (if t' is set to true, then x must be assigned to true; if f is set to true, then x must be assigned to false). In order to assign x to true, set f' to false. In order to assign x to false, set t to false. If neither of the t, t', f, or f' variables are set, then the solution is not unique.

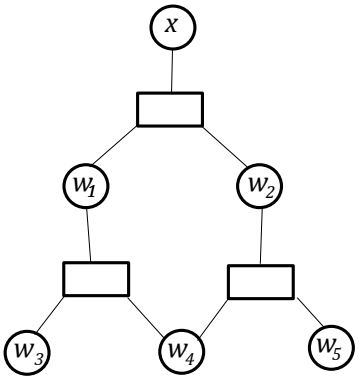


Figure 3: Assign gadget for PLANAR 1-IN-3 SAT. Each box represents a 1-IN-3 SAT clause with variables corresponding to the three lines connected to the box. x is set to true by setting w_4 to true. x is set to false by setting w_1 or w_2 to true (but not both). If none of the w_i in the clue gadget is assigned, there exists more than one satisfying assignment.

Lemma 8 UQ PLANAR 1-IN-3 SAT *is* Σ_2^P*-hard.*

Proof: The reduction is from UQSAT. We use the parsimonious reduction from SAT to PLANAR 1-IN-3 SAT found in [4]. □

Lemma 9 FCP PLANAR 3SAT *is* Σ_2^P*-complete.*

Proof: The reduction is from UQ3SAT. We use the reduction from [6] showing PLANAR 3SAT is NP-complete. The only change is that we introduce clauses to form an assign gadget as shown in Figure 2. The assign gadget guarantees variables with existential quantifiers in UQ3SAT are given clues. Specifically, on instances of UQ3SAT that admit a solution, there is a unique assignment of the variables in the gadget that produce a valid solution. If UQ3SAT is not satisfiable, on the other hand, then no assignment of the variables in the assign gadget will make it satisfiable. □

Lemma 10 FCP PLANAR 1-IN-3 SAT *is* Σ_2^P*-complete.*

Proof: The reduction is from UQ PLANAR 1-IN-3 SAT. We use the assign gadget from Figure 3 for each variable with an existential quantifier. This assign gadget is designed so that any clue to the problem must resolve an ambiguity within the gadget, forcing every clue to assign a value to each existentially quantified variable. If the instance is satisfiable, then a unique solution exists and the gadget has a satisfying assignment implied by the assignment to one of the clues. If the instance is not satisfiable, then no assignment of the gadget variables will satisfy the constraints. Therefore, even though FCP PLANAR 1-IN-3 SAT can give clues for any variable, because of the assign gadgets, clues will only correspond to variables with existential quantifiers in UQ PLANAR 1-IN-3 SAT.

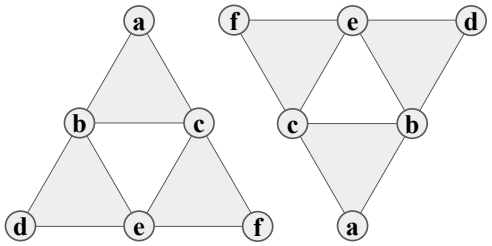


Figure 4: Gadget showing the T and F partitions for the two-way join from [3].

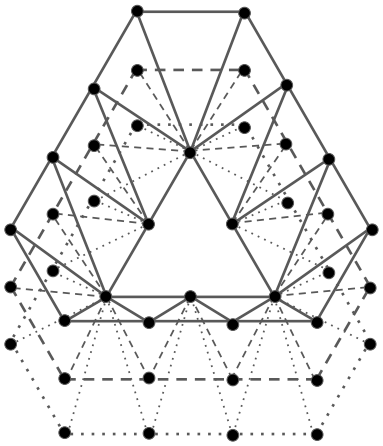


Figure 5: Three-way join gadget from [3]. Note that one of the three graphs must be partitioned in order to obtain a triangle partition of the join.

□

Lemma 11 FCP 1-IN-3 SAT *is* Σ_2^P*-complete.*

Proof: This follow immediately from Σ_2^P-hardness of FCP PLANAR 1-IN-3 SAT. □

4 FCP Versions of NP-hard Puzzles

In this subsection we show that the FCP versions of popular puzzles are Σ_2^P-complete. It is worth noting that all problems considered in this section are in NP, so their FCP versions are in Σ_2^P due to Theorem 4. Therefore, we focus on obtaining Σ_2^P-hardness reductions.

4.1 Triangle Partition

The Triangle Partition problem is the following:

Given an undirected graph $G = (V, E)$, can we partition E into disjoint triangles?

Holyer [3] showed this problem is NP-complete. Since we will make non-trivial modifications to Holyer's proof, we briefly review his proof at a high level. [3] constructs a graph H

which has the property that it can be partitioned in one of two ways, which he refers to as T-partition and F-partition. He describes how to join vertices and edges together to achieve the following effects:

- In Figure 2 of [3] (reproduced as Figure 4 here), two graphs can be associated with a *two-way join*. We abbreviate the join as a tuple (a, b) where a and b are two copies of H. Note that the order here matters. This association of vertices between a and b enforce that if a is T-partitioned, then b is T-partitioned. Additionally, via a simple transformation, which we refer to as $(a, b)'$, we can enforce that if a is T-partitioned, then b is F-partitioned.
- In Figure 3 of [3] (reproduced as Figure 5 here), three graphs can be associated with a *three-way join*. We will abbreviate this join with a three-tuple (a, b, c) where a, b and c are copies of H, where order here does not matter. The three-way join has the effect that exactly one of the three graphs a, b, or c will be T-partitioned, and the other two will be F-partitioned.

Each variable and each literal is represented as a copy of H. If the graph corresponding to a variable is T-partitioned, then that variable would be assigned to true, and if the graph is F-partitioned, it is assigned to false. The three literals of a clause are joined with a three way join. A literal l_x and its corresponding variable x are joined with the two-way join (l_x, x) if l_x is positive, and $(l_x, x)'$ if l_x is negative.

In FCP Triangle Partition, we are given a graph as well as an integer k, and must decide whether there exists a fixing of k triangles such that the remaining graph has a unique triangle partition. We use elements of Holyer's reduction to show FCP Triangle Partition is Σ_2^P-complete. We use the notation as well as elements from the reduction in [3].

Theorem 12 *FCP Triangle Partition is Σ_2^P-complete.*

Proof: In order to show FCP Triangle Partition is Σ_2^P-hard, we reduce from FCP 1-IN-3 SAT. We use Holyer's reduction, which reduces 3SAT to Triangle Partition [3]. For the FCP setting, we can utilize gadgets from [3] to reduce from FCP 1-IN-3 SAT to FCP Triangle Partition. We utilize the gadget shown in Figure 4 applied twice. In particular, if the literal l_x appears positive, we apply the two-way join (l_x, x) and (x, l_x). Thus, x and l_x are either both T-partitioned, or both F-partitioned. Similarly, if l_x appears negative, we apply the join $(l_x, x)'$ and $(x, l_x)'$.

We utilize the three-way join (Figure 5) to implement clauses in the same way as [3]. The gadget enforces that exactly one of the three literals is T-partitioned and the other two are F-partitioned.

This completes the proof, as now a clue on variables in the 1-IN-3 SAT instance corresponds exactly to a clue of the same size on the variable gadgets in the Triangle Partition instance. □

4.2 Latin Squares

The Latin Squares problem is the following:

> Given an $n \times n$ grid, with some entries filled in with the numbers 1 to n, can we fill in the remainder of the grid with numbers from 1 to n so that no row or column repeats a number?

In the FCP version of this problem, we are given a partial latin square, as well as an integer k, and we must decide whether there exists k entries such that fixing these k entries would make the partial latin square uniquely solvable. Now, we show FCP Latin Squares is Σ_2^P-complete.

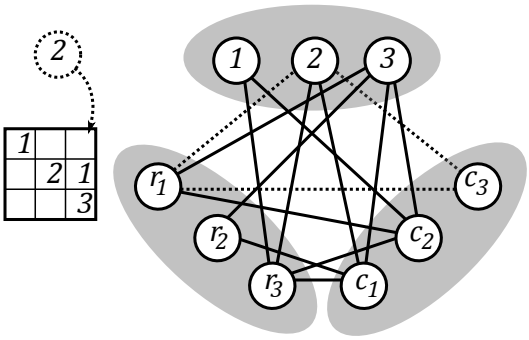


Figure 6: Example of a 3×3 partial latin square and its corresponding *defect* graph (see [1] for a formal definition). We highlight the correspondence between a triangle in the graph and an entry of the latin square. In [1], Colbourn shows that every tripartite graph is the defect graph of some partial latin square, and provides a polynomial time algorithm for constructing a partial latin square with the corresponding defect graph.

Theorem 13 *FCP Latin Squares is Σ_2^P-complete.*

Proof: We show hardness of FCP Latin Square via a reduction from FCP Triangle Partition of Tripartite Graphs, which is also Σ_2^P-complete following Theorem 12 and Theorem 2.1 of [1]. The authors of [1] introduce *defect graphs*, which are bijective mappings from partial Latin Squares to Tripartite Graphs. Each part of the tripartite graph corresponds to a set of size n: one part is the numbers 1 to n, another the rows 1 to n and the last part corresponds to the columns 1 to n. The edges on this graph are drawn to represent the filled in entries of the Latin Square instance. For instance, if the (i, j) entry of the Latin Square instance has the number c written on it, then the three edges connecting the vertices corresponding to the number c, the i-th row and the j-th vertex are present in the graph (see Figure 6 for a simplified example).

In order to show that this reduction is valid in the FCP setting we must show that k clues of an instance of the Triangle Partition of Tripartite Graphs problem with a unique solution corresponds to an instance of the Latin Square problem with k clues, and any instance of the latter with k clues can be implied by an instance of the former with at most k clues. The authors of [1] show that every tripartite graph can be transformed into a partial Latin Square which can be completed if and only if the tripartite graph can be partitioned into triangles. This, combined with the fact that each triangle corresponds bijectively to a filled entry of the Latin Square, proves the first direction. The other direction follows by a similar reasoning. Any instance of the Latin Square problem with k clues can be derived by an instance of the Triangle Partition problem with exactly k triangles, namely those that arise from constructing the defect graph of the Latin Square instance. □

4.3 Sudoku

The Sudoku problem is as follows:

> Given an $n^2 \times n^2$ grid, divided into n^2 subgrids of size $n \times n$, can we fill in the grid with numbers from $[n^2]$ so that no row, column, or any of the n^2 subgrids contain any repeated numbers?

We will show the FCP problem is Σ_2^P-complete by reduction from FCP Latin Squares. The Latin Square is embedded into the Sudoku puzzle. The columns of the Latin Square are mapped to the parts of columns of the Sudoku. In the mapping, the values are scaled by n.

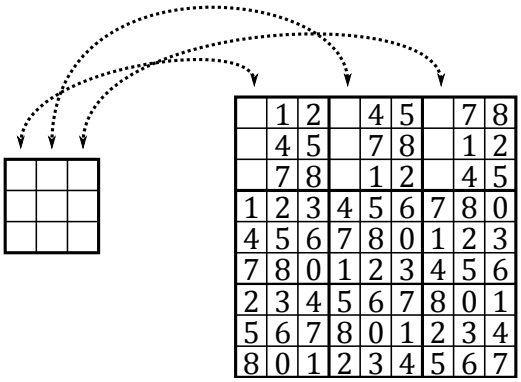


Figure 7: Reduction from Latin Square to Sudoku for $n = 3$ from [9].

Lemma 14 *FCP Sudoku is Σ_2^P-complete.*

Proof: We note the same reduction as [9] will serve as a reduction from FCP Latin Square to FCP Sudoku. Recall that an instance of the Sudoku problem of size n is a grid of size $n^2 \times n^2$, with n^2 squares of size $n \times n$ distinguished. For the construction $n^4 - n^2$ of the entries in the Sudoku puzzle are pre-filled and this depends only on n. Moreover, the pre-filled entries are consistent with a solvable Sudoku puzzle. The missing entries in the pre-filled Sudoku are the first column of each of the n squares in the first "row" of $n \times n$ squares. The pre-filled grid is constructed so that the missing entries must be multiples of n. If $c \in [1, n]$ is the entry on the i-th row and j-th column of the Latin Square instance, the reduction will write $c \cdot n$ on the i-th row and $(n \cdot (j-1) + 1)$-th column of the Sudoku instance. Figure 7 shows the case when $n = 3$.

In order to show this is a valid FCP reduction, we must show that an instance with k clues and a unique solution to the Latin Square problem maps to an instance with k clues and a unique solution for the Sudoku problem, and that any Sudoku problem with k clues and a unique solution can be derived from a Latin Square instance with at most k clues. The former follows from the fact that any entry in the Latin Square problem maps to exactly one missing entry in the Sudoku puzzle, and any filling of the Latin Square entry will give a value for its corresponding Sudoku entry. This implies that a set of k clues will remain of size k and that any entry implied in the Latin Square problem will have a matching entry implied in the Sudoku problem. The latter follows from the fact that k clues on the Sudoku puzzle will map back to k clues in the Latin Square problem (since the mapping between entries is bijective). This also implies that any set of k clues to the Sudoku puzzle is implied by a set of at most k clues in the Latin Square problem. □

4.4 Shakashaka

Shakashaka is a pencil-and-paper puzzle published by Nikoli. A Shakashaka puzzle consists of a rectangular grid of black and white squares. Black squares either contain an integer from $\{0, 1, 2, 3, 4\}$ or can be left blank. Each white square is filled in with a black triangle or is left blank. The filled-in squares become half-black with right isosceles triangles in any of four possible directions (referred to as b/w squares in [2]). The resulting configuration must satisfy the following constraints: each black square with a number c must have c adjacent black isosceles triangles, and the remaining white area must be partitioned into (possibly rotated) rectangles.

The problem is shown to be NP-complete in [2]. The same reduction will show that FCP Shakashaka is Σ_2^P-complete.

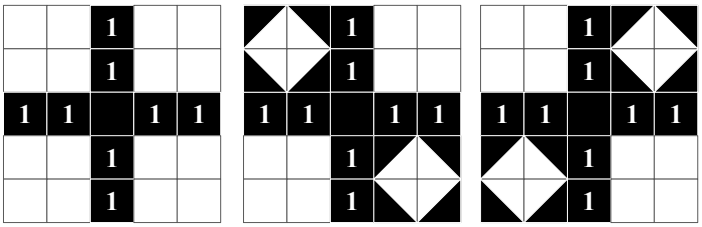


Figure 8: Shakashaka variable gadget (left) with both valid fillings corresponding to True and False assignments [2].

Lemma 15 *FCP Shakashaka is Σ_2^P-complete.*

Proof: We use the reduction from [2] to reduce from FCP Planar 3SAT. The reduction works by building "variable", "wire", "split", "corner", "parity", and "clause" gadgets in order to represent the graph of the planar 3SAT instance as an instance of Shakashaka. For each variable in the original planar 3SAT graph, we have a corresponding Shakashaka variable gadget, and similarly for each clause. True / false variable assignments are represented by picking between two possible Shakashaka square fillings within the variable gadget (see Figure 8). The wire gadget given in [2] (see Figure 9) propagates these true / false settings, with help from the split, corner, and parity gadgets that allow for splitting the wire into two wires, turning the wire, and adjusting the position of the wire (to allow for negations), respectively. The wires connect variable gadgets to clause gadgets, which are satisfied exactly when at least one incoming wire propagates a "true" signal.

For this reduction, we must prove two statements. First, if there exists an assignment of k clues for a planar 3SAT instance (i.e. truth assignments for k variables) that uniquely determines a 3SAT solution, we can identify an assignment of k Shakashaka squares that gives a unique Shakashaka solution. Second, if there exists an assignment of k Shakashaka squares that gives a unique Shakashaka solution, then there is an assignment of *at most* k variables in the corresponding planar 3SAT instance that gives a unique solution.

The first observation is that every white square in the entire Shakashaka instance can be implied by an assignment within the variable gadget. This can be confirmed by looking at each gadget in [2], and noting that every white square exists solely to propagate a true/false assignment from some variable gadget. Thus, we can assume without loss of generality that any clues given in the Shakashaka instance are given within the variable gadgets, and nowhere else. The next observation is that setting a variable gadget to true or false requires assigning only one black/white square. As the two configurations of the variable gadgets have no white squares in common, the rest of the variable gadget is immediately determined.

Thus, the first statement is satisfied, since an assignment of k variables in planar 3SAT can be simulated in Shakashaka by assigning the same truth values to the corresponding k variable gadgets. This is done by giving a single white square assignment in each of those variable gadgets. Furthermore, the second statement holds, since any assignment of k Shakashaka squares can be implied by setting k squares within variable gadgets. These settings can be immediately interpreted as at most k variable assignments for planar 3SAT. □

4.5 Akari

Akari, or Light Up, is a pencil-and-paper puzzle consisting of a grid with light and dark squares. Dark squares may be labelled with integers from $\{0, 1, 2, 3, 4\}$. The player places

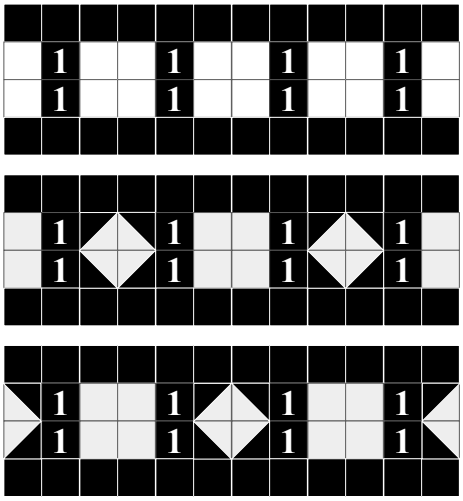


Figure 9: Shakashaka wire (top) with both valid fillings corresponding to transmitting True and False assignments [2].

lightbulbs on light squares which illuminate all light squares in the four directions around the lightbulb (although light stops upon a dark square). Equivalently, a square is lit if there exists at least one light bulb connected to it by a straight line of light squares. The goal is to place lightbulbs so that every light square is lit, no two light bulbs illuminate each other, and every numbered dark square has exactly that many light bulbs adjacent to it.

This problem is known to be NP-complete. We show that its FCP version is Σ_2-complete.

Lemma 16 *FCP Akari is Σ_2^P-complete.*

Proof:

In order to show FCP Akari is Σ_2^P-complete, we reduce from UQSAT. We use the reduction from Circuit SAT to Akari from [7], which details how to transform an arbitrary boolean circuit into an Akari instance. This is done by constructing "wire" gadgets that allow for exactly two types of lightbulb settings, corresponding to true/false signals. In addition to the wires, there is "branch" gadget for splitting a signal, a "NOT" gadget for simulating a NOT gate, an "OR" gadget for simulating an OR gate, and a construction that allows for wires to cross. We note that we are simplifying the original construction; in the original paper, [7] give combination gates that perform multiple functions (such as outputting both the OR and the exclusive NOR), but also provide a gadget to "cap" outputs which allows for constructing the basic gates listed. The output wire of this construction is terminated in such a way that forces the wire signal to be true. Thus, a solution to the Akari instance exists if and only if the circuit is satisfiable.

Now, consider an instance of UQSAT. First, we represent this SAT instance as a physical boolean circuit. Following the reduction of [7], we then build a corresponding instance of Akari. At this point, we modify the Akari instance by adding an assign gadget at each input wire corresponding to a variable in UQSAT with an existential quantifier. The assign gadget is detailed in Figure 10, and the wire itself is given in Figure 11. The assign gadget is introduces independent ambiguity into the Akari instance that can be resolved with a single clue for each assign gadget. The clue then uniquely determines the truth setting of the corresponding variable.

Let k be the number of variables with existential quantifiers (and hence the number of assign gadgets). If the UQSAT instance is satisfiable, then a correct setting of the existentially

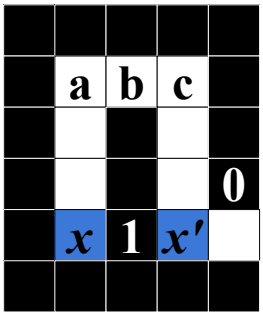


Figure 10: Akari assign gadget. If x or x' is lit, then there is ambiguity regarding placing the light in b or c, in the case of x, and a or b in the case of x'. Lighting up a means x' must be lit, and lighting up c means x must be lit.

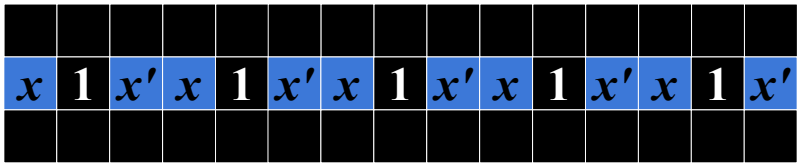


Figure 11: Akari wire gadget. Carries the value of the variable as x or x' [7].

quantified variables can be given as k clues to the Akari assign gadgets. By definition of UQSAT, there is a unique setting for the remaining variables, and the same holds true in the Akari instance. In the other direction, if there is a setting of k clues in Akari that identifies a unique solution, those k clues must be for the k variables with assign gadgets. Otherwise, the ambiguity in the assign gadgets will not be resolved. The fact that the remaining variables have a unique solution means that the instance of UQSAT is satisfiable.

□

5 FCP Versions of Easy Problems

In this section, we consider the complexity of the FCP versions of various problems in P.

5.1 Problems Closed under Cluing

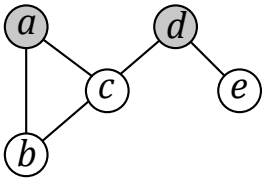
Definition 17 *We call a problem* closed under cluing *if plugging any partial solution into a problem instance results in another instance of the same problem.*

For example, 2SAT is closed under cluing since plugging in values for some of the variables will result in another instance of 2SAT.

Proposition 1 *If A is a problem closed under cluing in P, then* FCP $A \in$ NP.

Proof:

We give the following nondeterministic polynomial time algorithm. FCP A asks us to find a clue set of size at most k. First, nondeterministically pick a clue set of size k. Note that



$$(\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \\ \wedge(\neg b \vee \neg c) \wedge (\neg c \vee \neg d) \\ \wedge(\neg d \vee \neg e) \tag{1}$$

Figure 12: Reduction from Minimum Independent Dominating Set to FCP 2SAT.

the size of the certificate is polynomial in that of the input. Then for each string character in the solution that has not been assigned a value, try all possible settings of that character and accept if only one setting of that character produces a solvable problem. This step relies on closedness, as we use the fact that the problems that result from setting certain characters of the solution are still solvable in polynomial time.

If for any string character we find that multiple settings give a solvable problem, then our clue set does not give a unique solution. If no setting gives a solvable problem, then our clue set is not valid since no solution is possible.

□

5.2 FCP 2SAT

Proposition 1 states that FCP 2SAT $\in$ NP. Here, we show that this problem is in fact NP-complete.

Theorem 18 *FCP* 2SAT *is NP-complete.*

We reduce from the NP-hard problem, Minimum Independent Dominating Set (MIDS), which asks whether a graph has an independent dominating set of size at most k.

> Given a graph $G = (V, E)$, does there exist $S \subset V$, with $|S| \leq k$ such that, for all $v \in V$, either $v \in S$ or $(u, v) \in E$, $u \in S$, and $u, v \in S$ implies that there is no edge between u and v.

We transform G into a formula ϕ. For each edge (u, v), we add the constraint $(\neg u \vee \neg v)$ to the 2SAT formula. The following two lemmas prove that this reduction is correct.

Lemma 19 *Suppose G contains an independent dominating set of size at most k, then there exists a clue of the* 2SAT *formula containing at most k variables.*

Proof: Suppose $S \subset V$ is an independent dominating set of size k. Then for each $s \in S$, assign s to true. We claim the formula is uniquely satisfiable.

If x is some variable in ϕ, then x corresponds to some node in G. Since S is a dominating set, let $s \in S$ be a neighbor of x. Then $(\neg x \vee \neg s)$ is a clause in ϕ. For ϕ to be satisfied, x must be false.

Since S is independent, we have no false clauses. □

Lemma 20 *Suppose ϕ contains a clue of size k. Then G contains an independent dominating set of size at most k.*

Proof:

Let C_T and C_F be the set of variables set to true and false, respectively, in the clue for ϕ. Without loss of generality, we may force C_F to be empty. For any variable x in C_F, consider a clause $(\neg x \vee \neg y)$ in ϕ, where y is a new variable. We can give a clue of identical size by removing x from C_F and including y in C_T.

Also, we note that all variables assigned to true in the satisfying assignment must be in C_T. If a variable x set to true is not in C_T, uniqueness is violated since setting x to false gives another satisfying assignment.

Let $S = C_T$. S must be an independent set because ϕ is satisfiable. In addition, if $v \in G - S$, then since the satisfying assignment to ϕ is unique, v contains a neighbor whose variable is set to true. So S is dominating. □

5.3 FCP versions of other easy problems

Consider the following language:

$$L = \{\phi' = (\phi \wedge \overline{z}) \vee z | \phi \text{ is a Boolean formula and } z \text{ does not appear in } \phi\}.$$

Note that if we ask whether ϕ' is satisfiable, the answer is trivially yes. So $L \in \mathsf{P}$. FCP L asks whether there exists a setting of k variables to ϕ' which makes it uniquely satisfiable.

Proposition 2 *FCP L is Σ_2^P-complete.*

Proof: We reduce from FCP SAT. If ϕ has a clue of size at most k with a unique solution, then $\phi' = (\phi \wedge \overline{z}) \vee z$ has a partial assignment of size at most $k+1$ with a unique solution. We simply add the assignment for variable z.

If ϕ has no satisfying assignment, then ϕ' requires at least $k+1$ assigned variables for a partial assignment with a unique solution. Let x be a particular satisfying assignment to ϕ' that requires more than k assigned variables. If ϕ has a partial satisfying assignment $\mathbf{x}$ and $z \in \mathbf{x}$, then $\mathbf{x}$ needs at least n assigned variables to be specified. If $\overline{z} \in \mathbf{x}$, then we must specify $\overline{z}$ in the clue, and by assumption we must specify more than k variables. Therefore, if ϕ needs a clue of size greater than k, ϕ' will need a clue of size greater than $k+1$. □

Acknowledgements

This work began as a final project for MIT's Algorithmic Lower Bounds class (6.890) taught by Erik Demaine in Fall 2014. Thanks to Sarah Eisenstat and Jayson Lynch for valuable discussion and insight. This work is supported in part by the NSF Graduate Research Fellowship under Grant No. DGE-16-44869.

References

[1] Charles J Colbourn. The complexity of completing partial Latin squares. *Discrete Applied Mathematics*, 8(1):25–30, 1984.

[2] Erik D Demaine, Yoshio Okamoto, Ryuhei Uehara, and Uno Yushi. Computational complexity and an integer programming model of shakashaka. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 97(6):1213–1219, 2014.

[3] Ian Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.

[4] Harry B Hunt III, Madhav V Marathe, Venkatesh Radhakrishnan, and Richard E Stearns. The complexity of planar counting problems. *SIAM Journal on Computing*, 27(4):1142–1167, 1998.

[5] Philippe Laroche. Planar 1-in-3 satisfiability is NP-complete. *Comptes rendus de l'academie des sciences serie I-Mathematique*, 316(4):389–392, 1993.

[6] David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[7] Brandon McPhail. Light up is NP-complete. http://www.mountainvistasoft.com/docs/lightup-is-np-complete.pdf, 2005.

[8] Takahiro Seta. The complexities of puzzles, cross sum and their another solution problems (ASP). Senior thesis, Univ. of Tokyo, Dept. of Information Science, Tokyo, Japan, Feb 2001.

[9] Yato Takayuki and Seta Takahiro. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

[10] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, October 1991.

[11] Nobuhisa Ueda and Tadaaki Nagao. NP-completeness results for nonogram via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan, 1996.

[12] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[13] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[14] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(C):85–93, 1986.