修士論文

覆面算・虫食い算の計算複雑性に関する研究

入野 耀太

主指導教員 上原 隆平

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和 7 年 3 月

Abstract

In this paper, we investigate the computational complexity of two types of pencil puzzles: cryptarithms and arithmetical restorations.

A cryptarithm is a puzzle in which the digits of an arithmetic expression are replaced by letters, and the objective is to assign distinct digits to each letter in a way that the expression becomes valid.

In contrast, arithmetical restorations involve an arithmetic expression with certain digits replaced by blanks, where the goal is to assign appropriate digits to these blanks to complete the expression.

Previous research has examined the complexity of both puzzles under several conditions. Eppstein demonstrated the NP-completeness of cryptarithms for the addition of two numbers, while Matsui proved NP-completeness for arithmetical restorations involving multiplication of two numbers.

In this work, we extend these results by showing that cryptarithms are also NP-complete for multiplication, subtraction, and division of two numbers.

Furthermore, we develop a linear-time algorithm for the number of digits such as counting up the total number of solutions using dynamic programming for solving arithmetical restorations in cases of addition and subtraction.

Extending this result, we develop a polynomial-time algorithm for k and the number of digits of k-number addition arithmetical restorations. We also developed a linear-time algorithm for the number of digits of recently proposed double arithmetical restorations. In addition, we prove NP-completeness for arithmetical restorations involving division.

These results provide a comprehensive analysis of the computational complexity of all four basic arithmetic operations for both cryptarithms and arithmetical restorations.

目次

図目次

1　はじめに

1.1　研究の背景・目的

理論計算機科学の根底には，計算とはなにか，計算可能な関数とはなにかという問いがある．直感的には，一連の基本的な操作を行うことで問題の解を求めることが計算である．では逆に，計算不可能な関数は存在するのか，存在したとして計算なしにそれを認識することができるのだろうか．この問題は 1930 年代から考えられていたが，クリーネ，チャーチ，チューリングなどによって研究され，計算可能関数の概念が提唱されて決着がつけられた．それ以来，計算可能な関数の集合についての理解を深めるために多くの研究がなされてきた．

計算可能な関数の集合を理解するためのアプローチとして，数学の分野では帰納的関数論についての研究が行われている．また，計算機科学の分野では，計算可能な関数の集合を理解するためにチューリングマシンモデルに基づいて，計算の複雑さに関する研究が行われている．特に，様々な問題の難しさを特定するために，多くの計算複雑性クラスが導入されている．

そのほかのアプローチとして，一般的なパズルやゲームを研究することが挙げられる．パズルやゲームといった枠組みは計算の本質と深く関わりを持つ．パズルやゲームで使われるルールを計算の基本操作の集合と考えれば，それらは計算モデルとみなすことができる．一般に，パズルやゲームはチューリングマシンと比べて直感的という意味で単純な構造を持っている．そのため，パズルやゲームについて解析することにより計算についての理解を深めることができる．

実際，非決定性チューリングマシンの概念はある種のパズル，例えばペンシルパズルに自然な形で対応する．また，多くのペンシルパズルは NP 完全である [1, 2]．一般にペンシルパズルでは埋めるべきマスがあって，それを順々に埋めていくということを行う．正解にたどり着くには正しくマスを埋め続ける，つまり正しい選択肢を選び続ける必要があり，間違えば正解にたどり着くことはない．一方で，ペンシルパズルのマスを埋めたときに，それが答えとしてあっているかどうか確かめることは簡単である．

これまでに，ペグソリティア [3]，数独 [4, 5]，カックロ [6]，ひとりにしてくれ [7] など様々なパズルについて研究されてきており，多くのパズルに対して NP 完全性が示されている [1, 2, 5]．それぞれのパズルは単純な構造とルールを持っているので，NP 完全問題の難しさを直感的に理解することができる．つまり，これらの単純なパズルに共通する性質が，計算複雑性クラス NP の難しさの本質であると考えられる．

パズル以外の問題を含め膨大な数の問題が NP 完全であることが示されている．また，新しい問題 A の NP 完全性を示したいとき，既に NP 完全性が証明された問題の中から A に似た問題 B を見つけて困難性を比較することによって，NP 完全性を示せる場合がある．具体的には，問題 B のどのような出題例 Y に対しても，Y を解くことと同値な問題 A の出題例 X を構成できることを示すことで，問題 A を解くことが問題 B を解くことと同等以上に難しいことを証明できる．この方法は多項式時間還元と呼ばれ，特定の問題の NP 完全性を示す際によく用いられる．よって，問題の NP 完全性を示すことは学術的・実用的に重要である．これに関連して，どの問題がどの計算複雑性クラスに属するのか明らかにすることには価値がある．

前述の NP 完全性の研究が「そのパズルに解が存在するかどうか」に注目していたのに対して，「そのパズルが唯一解を持つかどうか」に注目した研究も近年盛んに行われている．例えば，ASP という計算複雑性クラスや FCP という枠組みが提案されている．前者は，Another Solution Problem の略である．これは，ある問題のインスタンスとその解が与えられたときに，別の解を求める問題であり，Seta によって提案された [6]．後者は，Fewest Clues Problem の略である．これは，ある問題のインスタンスが与えられたとき，事前にいくつかのヒントを与えて解を制限することで，解を唯一にできるかどうかという問題を定義する枠組みである [8]．

これまで，著名な多くのパズルについては，その計算複雑性が明らかにされてきた．しかし，覆面算・虫食い算についての研究は限定的なものにとどまっている [2, 5]．具体的には，1987 年に Eppstein が足し算の覆面算 [9]，2013 年に Matsui が掛け算の虫食い算 [10] の計算複雑性を明らかにしている．これまでに覆面算・虫食い算の研究が広く行われてこなかった理由のひとつとして，理論計算機科学における研究対象の流行り廃りのなかで取り残されてしまった可能性が考えられる．実際，1990 年代には囲碁・将棋・オセロ・ボードゲームの計算量が活発に研究され，その後には数独やテトリスといったパズルが研究対象になるなど，その時々でさまざま流行がある [1]．

本研究の目的は，覆面算・虫食い算の計算複雑性について個別に解析を行うことによって，それぞれの四則演算に関する覆面算・虫食い算が属する計算複雑性クラスを明らかにすることである．

1.2　覆面算・虫食い算について

覆面算とは与えられた各桁が記号で構成される筆算の計算式に対し，異なる記号に異なる一桁の数字を割り当てるパズルである．ただし，最上位桁の記号に 0 を割り当てることは禁止されている．また，このパズルの制約から n 進数の覆面算に含まれる記号の種類は高々 n 種類である．

2 数の加算に関する覆面算の一例：SEND + MORE = MONEY と，その答えを図 1 に示す．この有名な覆面算は，イギリスの雑誌 Strand Magazine の 1924 年 7 月号に Dudeney が発表したものだが，考案者が本人かどうかは，わかっていない [11].

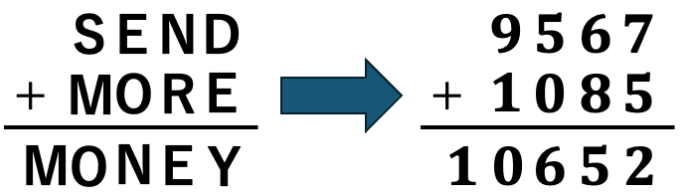

図 1: 覆面算 SEND + MORE = MONEY とその解答

虫食い算とは所々が空欄に置き換えられた筆算を成立させるように，空欄に一桁の数字を割り当てるパズルである．ただし，最上位桁の空欄に 0 を割り当てることは禁止されている．

2 数の除算に関する虫食い算の一例として，「孤独の 7」と呼ばれている除算の虫食い算と，その答えを図 2 に示す．この虫食い算は，Odlling によるもので Strand Magazine の 1922 年 11 月号の 439 ページに掲載されたのが初出といわれている [1, 12].

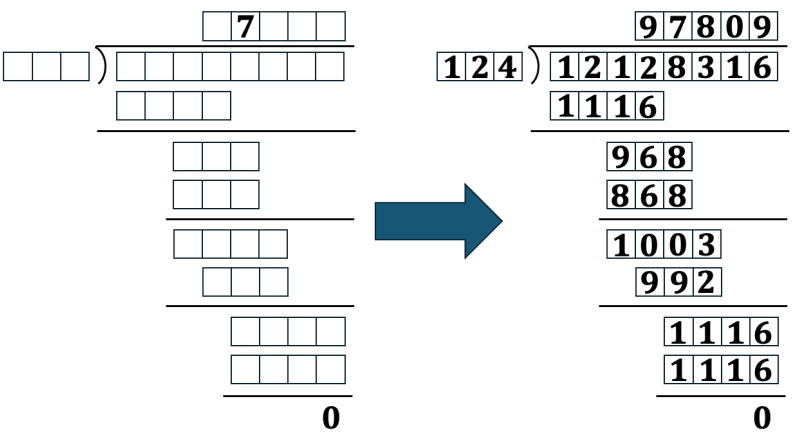

図 2: 虫食い算：「孤独の 7」とその解答

以降，与えられた覆面算と虫食い算が解を持つかどうかを判定する問題を，単に覆面算・虫食い算と呼ぶことにする．

覆面算と虫食い算はそれぞれ，いくつかの設定下で計算量の解析が行われている．Eppstein は，n 進数 m 桁の 2 数の加算の覆面算が NP 完全であることを 3SAT からの還元によって示した [9]．また，Matsui は，n 進数 m 桁の 2 数の乗算の虫食い算が NP 完全であることを Monotone 1-in-3 3SAT からの還元によって示した [10]．

1.3 本研究の主結果

本研究では，2 数の乗算・減算・除算に関する覆面算および，2 数の除算に関する虫食い算が NP 完全であることを明らかにし，2 数の加算・減算に関する虫食い算には桁数に関する線形時間アルゴリズムが存在することを示した（表 1）．また，2 数の加算の虫食い算を k 数の虫食い算に拡張した場合については，k と桁数に関する多項式時間アルゴリズムが存在することを示した．さらに，虫食い算の拡張として近年提案された「2 重虫食い算」というパズルについて，加算の 2 重虫食い算にその桁数に関する線形時間アルゴリズムが存在することを示した．

表 1: 各演算に関する覆面算・虫食い算の計算複雑性（太字は本研究の結果）

四則演算	覆面算	虫食い算
加算	NP 完全 [9]	$\in \mathbf{P}$
減算	**NP 完全**	$\in \mathbf{P}$
乗算	**NP 完全**	NP 完全 [10]
除算	**NP 完全**	**NP 完全**

1.4 本論文の構成

本論文の構成を説明する．まず，第 2 章では，乗算・減算・除算の覆面算の NP 完全性を証明する．次に，第 3 章では，加算・減算の虫食い算，k 数の加算の虫食い算のアルゴリズムを示す．その後，除算の虫食い算の NP 完全性を証明する．第 4 章では，加算の虫食い算の拡張である 2 重虫食い算のアルゴリズムを示す．最後に，第 5 章では，結語として本研究のまとめと今後の展望を述べる．

2　覆面算

本章では，Eppstein によって NP 完全性が示されている加算の覆面算 [9] から乗算の覆面算に還元することによって，乗算の覆面算が NP 完全であることを証明する．次に，加算の覆面算から減算の覆面算に還元することによって，減算の覆面算が NP 完全であることを証明する．その後，減算の覆面算から除算の覆面算に還元することによって，除算の覆面算が NP 完全であることを示す．

2.1　乗算の覆面算

本節では，乗算の覆面算の定式化と定義を行った後，乗算の覆面算の困難性の証明に利用する Eppstein による加算の覆面算の結果 [9] について説明する．最後に，乗算の覆面算が NP 完全であることを証明する．

2.1.1　問題の定式化と定義

乗算の覆面算を以下のように定式化する．

定義 2.1. 図 3 のように，$p \geq q$ に対して，n 進数 p 桁の数と q 桁の数の乗算に関する覆面算 $Y_\times$ は，$q+3$ 行の文字列 S_i $(i = 1, \ldots, q+3)$ から構成される．ここで，各 S_i は $l_i \in \{t \in \mathbb{N} \mid 1 \leq t \leq pq+1\}$ 個の文字の列であり，$l_1 = p, l_2 = q$ である．また，各 S_i の最上位桁に対応する文字に 0 を割り当てることは禁止されているものとする．ただし，$l_i = 1$ のときに限り，S_i に含まれる最上位桁に対応する文字に 0 を割り当てても良いものとする．

次に，覆面算 $Y_\times$ の解を次のように定義する．

定義 2.2. 覆面算 $Y_\times$ の各列 S_i に含まれる文字の集合を S とする．また，文字への割り当てを $s : S \to \{t \in \mathbb{N} \mid 0 \leq t \leq n-1\}$ とする．任意の $s_1, s_2 \in S$ に対して $s_1 \neq s_2 \Rightarrow s(s_1) \neq s(s_2)$ かつ，s の通りに文字に数字を割り当てたとき $Y_\times$ が計算式として成立するとき，s は覆面算 $Y_\times$ の解である．

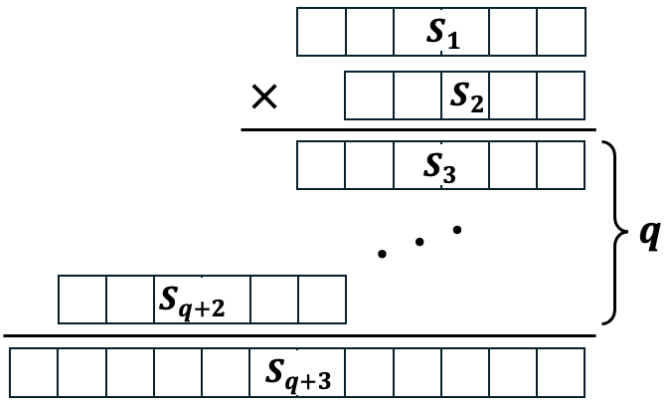

図 3: 定式化された乗算に関する覆面算

2.1.2 乗算の覆面算に関する結果

乗算の覆面算について次が成立する：

定理 2.3. 2 数の乗算に関する覆面算は NP 完全である．

この定理の証明は 2 .1.4 節で，加算の覆面算からの帰着によって行う．

2.1.3 既存研究の手法と結果

Eppstein により 2 数の加算の覆面算について，次が成立することが示されている [9].

補題 2.4. n 進数 m 桁の 2 数の加算の覆面算は NP 完全である．

証明は 3-SAT からの還元によって行われている．3-SAT は以下の問題である [13]：

入力： ちょうど 3 つの論理変数を含む節からなる CNF 論理式（3-CNF 論理式）．
問題： 与えられた CNF 論理式を充足させる，論理変数への割り当てが存在するかどうかを判定する．

Eppstein による証明では，与えられた 3-CNF 論理式に対応する覆面算を図 4, 5, 6 に示す 3 種類のガジェット，定数ガジェット C，変数ガジェット $G_V(i, v)$，節ガジェット $G_C(i, v_a, v_b, v_c)$ を組み合わせることで構成している．

定数ガジェット C は覆面算に含まれる記号 c_0, c_1 の値を，それぞれ 0 と 1 に確定する役割を持っている．

また，変数ガジェットと節ガジェットは，それぞれ 3-CNF 論理式に含まれる変数と節をシミュレートするガジェットであり，これらのガジェットに含まれる添字が i の文字は，この文字を含むガジェットの中でのみ役割を持つものである．

変数ガジェット $G_V(i, v)$ に含まれる $(v \mod 4)$ の値は 0，1 のどちらかであり，それぞれ 3-CNF 論理式に含まれる変数 x_v の値が偽，真に対応する．また，$(\bar{v_i} \mod 4)$ の値は $1 - (v \mod 4)$ であり，$\bar{x_v}$ の値に対応する．

節ガジェット $G_C(i, v_a, v_b, v_c)$ に含まれる，v_a, v_b, v_c は 3-CNF 論理式のある節に含まれる 3 つの変数 $x_{v_a}, x_{v_b}, x_{v_c}$ に対応する．ここで，節ガジェットの構造から，中に含まれる t_i について，$(t_i \mod 4)$ の値が 1 かつ $t_i = v_a + v_b + v_c$ が成立する．このことから，v_a, v_b, v_c のうち少なくとも 1 つは 4 で割ったときの余りが 1 のときに限り節ガジェット $G_C(i, v_a, v_b, v_c)$ が覆面算として成立する．

$$
\begin{array}{ccc}
c_0 & p & c_0 \\
c_0 & p & c_0 \\
\hline
c_1 & q & c_0
\end{array}
$$

図 4: 定数ガジェット C

$$
\begin{array}{cccccccccccc}
d_i & c_0 & c_1 & y_i & c_0 & c_i & y_i & c_0 & b_i & y_i & c_0 & a_i & c_0 \\
e_i & c_0 & d_i & y_i & c_0 & c_i & y_i & c_0 & b_i & y_i & c_0 & a_i & c_0 \\
\hline
\bar{v} & c_0 & e_i & z_i & c_0 & d_i & z_i & c_0 & v & z_i & c_0 & b_i & c_0
\end{array}
$$

図 5: 変数ガジェット $G_V(i, v)$

$$
\begin{array}{cccccccccccc}
u_{ab} & c_0 & v_a & c_0 & c_1 & r_i & c_0 & g_i & w_i & c_0 & f_i & c_0 \\
v_c & c_0 & v_b & c_0 & h_i & r_i & c_0 & g_i & w_i & c_0 & f_i & c_0 \\
\hline
t_i & c_0 & u_{ab} & c_0 & t_i & s_i & c_0 & h_i & x_i & c_0 & g_i & c_0
\end{array}
$$

図 6: 節ガジェット $G_C(i, v_a, v_b, v_c)$

2.1.4　定理 2.3 の証明

本節では，2 数の乗算の覆面算が NP 完全であることを，2 数の加算の覆面算から多項式時間で帰着することによって示す．

証明. まず，補題 2.4 より，2 数の加算の覆面算は NP 完全であり，3SAT に対応する 2 数の加算の覆面算を構築することで証明が行われている [9]．ここで，与えられた 3SAT に対応する 2 数の加算の覆面算を Y_+ とする．このとき，Y_+ に対応する，2 数の乗算の覆面算 $Y_\times$ を構成できることを示す．まず，変数ガジェット $G_V(i,v)$ と節ガジェット $G_C(i,v_a,v_b,v_c)$ を用いて構成した 2 数の加算の覆面算の 1 行目，2 行目，3 行目を，それぞれ G_1，G_2，G_3 とおく．次に，図 7 のように，文字 c_0，c_1 と文字の列 G_1，G_2，G_3 を組み合わせて 2 数の乗算の覆面算 $Y_\times$ を構成する．

ここで，Y_+ に対応する 3SAT の変数の個数を n'，節の個数を m' とすると，この Y_+ を元にして作った G_1, G_2, G_3 の長さは $13n' + 12m'$ である．よって，G_1, G_2, G_3 を組み合わせて構成した乗算の虫食い算 $Y_\times$ の中で最長の行の長さは $5 + 3(13n' + 12m')$ である．また，$Y_\times$ の行数は $6 + 13n' + 12m'$ である．したがって，$Y_\times$ のサイズは多項式で表現できるので，与えられた 3SAT の入力から $Y_\times$ を多項式時間で構築することが可能である．

次に，構築した乗算の虫食い算 $Y_\times$ が NP 完全であることを示す．まず，構成の仕方から c_0 の値は 0，c_1 の値は 1 に確定する．また，同図中の赤枠では，$G_V(i,v)$ と $G_C(i,v_a,v_b,v_c)$ のみで構成された 2 数の加算の覆面算 $G_1 + G_2 = G_3$ が成立する必要がある．したがって，このように構成した 2 数の乗算の覆面算 $Y_\times$ と，元の加算の覆面算 Y_+ は同一の解を持つ．よって，2 数の乗算の覆面算 $Y_\times$ の解を求めることは，2 数の加算の覆面算 Y_+ の解を求めることに帰着できる．以上より，3SAT が解を持つとき，かつそのときに限り $Y_\times$ が解を持つ．従って，2 数の乗算の覆面算は NP 完全である．　□

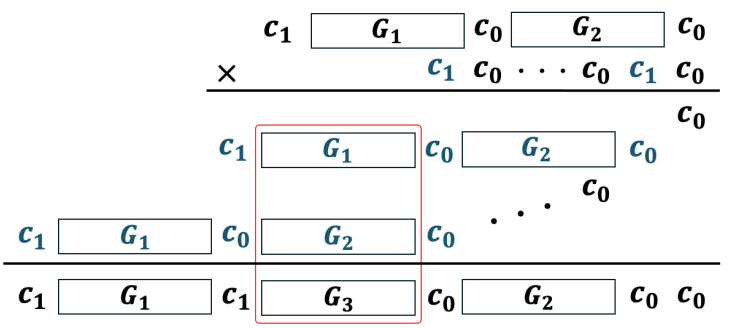

図 7: 加算の覆面算 Y_+ に対応する乗算の覆面算 $Y_\times$

2.2 減算の覆面算

減算の覆面算について以下が成立する：

定理 2.5. 2 数の減算に関する覆面算は NP 完全である．

証明. この問題が NP に属することは明らかであり，困難性の証明は加算の覆面算からの帰着によって行う．まず，加算の覆面算 Y_+ に対して，1 行目と 3 行目を入れ替えて，演算を $-$ にしたものを Y_- とする．次に，Y_- 内で，C，$G_V(i,v)$，$G_C(i,v_a,v_b,v_c)$ に対応する箇所を，それぞれ C'，$G'_V(i,v)$，$G'_C(i,v_a,v_b,v_c)$ とする．

ここで，Y_- は 3SAT に対応する加算の覆面算 Y_+ の 3 列目と 1 列目を入れ替えたものなので，Y_- のサイズは Y_+ と同じく 3SAT の変数と節の個数の多項式で表現できる．よって，与えられた 3SAT の入力から Y_- を多項式時間で構築することが可能である．

次に，構築した減算の覆面算 Y_- が NP 完全であることを示す．まず，C' については，その構造から c_0 と c_1 の値が，それぞれ 0 と 1 に確定される．次に，$G_V(i,v)$，$G_C(i,v_a,v_b,v_c)$ は図 8 に示す 4 パターンの加算の覆面算に分解できる．ここで，加算の覆面算を成立させる割り当てを s_+ とし，この通りに各文字に数字が割り当てられているとする．この割り当て s_+ のもとで，図 8 の加算の覆面算の 1 行目と 3 行目を入れ替えて作った減算の覆面算が成立することを示す．これが示せれば，減算の覆面算の解を求めることは，それに対応する加算の覆面算の解を求めることに帰着できることが証明できる．

図 8(a)，8(b) のパターンについて考える．このとき，加算において繰り上がりが生じていないので，それぞれ $x_1 < x_3$，$x_1 < x_2$ が成立する．よって，このパターンの 1 行目と 3 行目を入れ替えて作った減算の覆面算において，それぞれ $x_3 - x_2 = x_1$，$x_2 - x_1 = x_1$ が成立する．

図 8(c) のパターンの加算の覆面算で繰り上がりが生じていない場合について考える．このとき，$x_1 < x_2$ かつ $x_3 < x_4$ なので，この加算の覆面算を成立させる割り当ては，この覆面算を元にして作った減算の覆面算も成立させる．次に，元の加算の覆面算において繰り上がりが生じている場合，つまり $x_3 + x_3 \geq n$ について考える．このとき，$x_3 > x_4$ なので繰り下がりが生じる．図の 2 列目において元の加算の覆面算では $x_1 + x_1 = x_2 + 1$ が，3 列目においては $x_3 + x_3 = x_4 + n$ がそれぞれ成立しているため，差の覆面算では 2 列目で $x_2 - x_1 - 1 = x_1$ が，3 列目で $x_4 - x_3 - n = x_3$ それぞれ成立する．よって，繰り下がりが生じる場合でも割り当て s_+ で減算の覆面算が成立する．なお，図 8(d) のパターンについても同様に，加算の覆面算を成立させる割り当て s_+ が，この覆面算を元にして作った減算の覆面算を成立させることを示すことができる． □

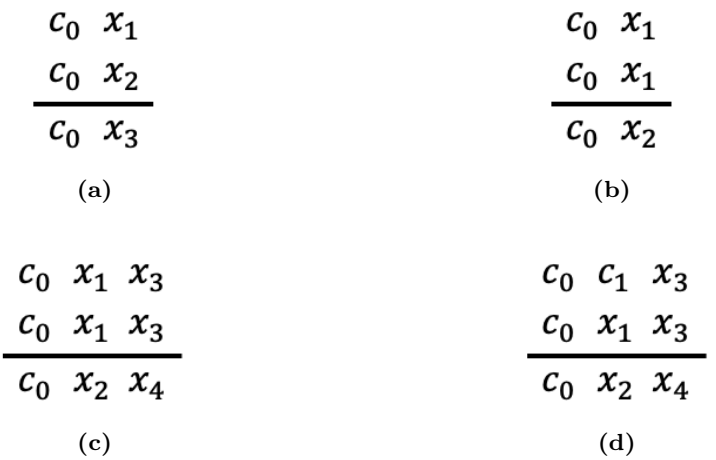

図 8: 加算の覆面算 $G_V(i,v)$，$G_C(i,v_a,v_b,v_c)$ の構成要素

2.3 除算の覆面算

除算の覆面算について次が成立する：

定理 2.6. n 進数 m 桁の 2 数の除算に関する覆面算は NP 完全である．

証明. この問題が NP に属することは明らかであり，困難性の証明は減算の覆面算からの帰着によって行う．まず，3.2 節で構成した 3SAT に対応する減算の覆面算 Y_- から，定数ガジェット C' に相当する部分を取り除いた覆面算の $1,2,3$ 行目を，それぞれ G'_1, G'_2, G'_3 とする．この G'_1, G'_2, G'_3 および文字 c_0，c_1 を図 7 のように組み合わせることで，3SAT に対応する 2 数の除算の覆面算 $Y_\div$ を構成する．

ここで，$Y_\div$ に対応する 3SAT に変数の個数を n'，節の個数を m' とすると，G'_1, G'_2, G'_3 の長さは $13n'+12m'$ であるので，$Y_\div$ に含まれる文字列の最大の長さは $5+2(13n'+12m')$ である．また，$Y_\div$ を構成する文字列は 9 個である．よって，$Y_\div$ のサイズは多項式時間で表現できるので，与えられた 3SAT の入力から $Y_\div$ を多項式時間で構築できる．

次に，構築した乗算の虫食い算 $Y_\div$ が NP 完全であることを示す．構成の仕方から c_0 の値は 0 に，c_1 の値は 1 に確定する．また，同図中の赤枠内では，$G'_V(i,v)$, $G'_C(i,v_a,v_b,v_c)$ のみで構成された 2 数の減算の覆面算 $G'_1 - G'_2 = G'_3$ が成立する必要がある．この減算の覆面算を成立させる割り当ては，$Y_\div$ を成立させるので，除算の覆面算 $Y_\div$ の解を求めることは，それに対応する減算の覆面算 Y_- の解を求めることに帰着できる．以上より，3SAT が解を持つとき，かつそのときに限り $Y_\div$ が解を持つため，2 数の除算の覆面算は NP 完全である．

□

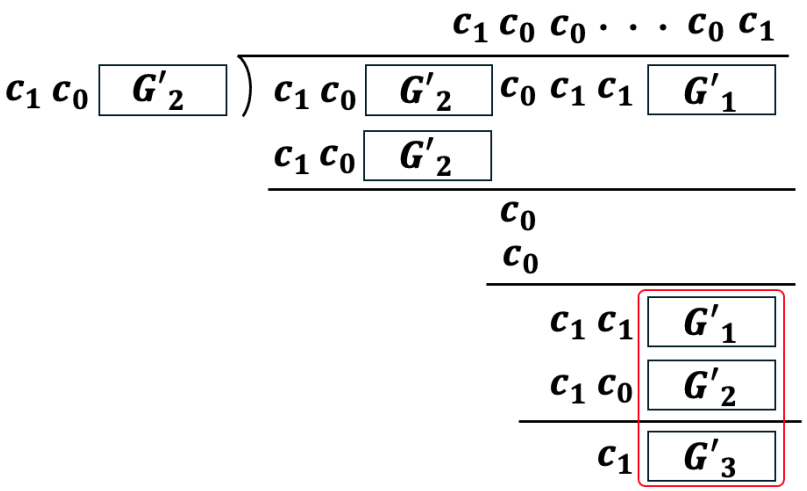

図 9: 減算の覆面算 Y_{-} に対応する除算の覆面算 $Y_{\div}$

3　虫食い算

本章では，加算および減算の虫食い算に関するアルゴリズムを示した後に，この結果を拡張することで k 数の加算の虫食い算に関するアルゴリズムを示す．次に，Matsui によって NP 完全性が示されている乗算の虫食い算 [10] から除算の虫食い算に還元することによって，除算の虫食い算が NP 完全であることを証明する．

3.1　加算および減算の虫食い算

本節では，加算・減算の虫食い算の定式化と定義を行った後，加算・減算の虫食い算の結果について説明する．次に，加算の虫食い算が解を持つ条件について述べる．最後に，加算・減算の虫食い算に対するアルゴリズムを構築し，その結果を拡張することで k 数の加算の虫食い算に関するアルゴリズムを示す．

3.1.1　問題の定式化と定義

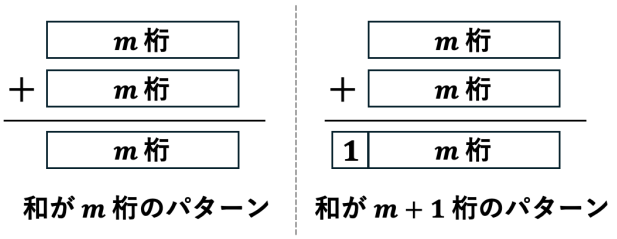

図 10: 2 数の加算に関する虫食い算の和のパターン

まず，加算の虫食い算を定義し，それを扱うための定式化について議論する．具体的には，加算の場合についてのみ定義するが，引き算も同様に定義する．

定義 3.1. n 進数 m 桁の 2 数の虫食い算のインスタンスを X_+ とし，X_+ は 2 つの加数とその和に対応する 3 行で構成されるものとする．2 つの加数は，それぞれ m 桁なので和は m 桁か $(m+1)$ 桁のどちらかである．後者の場合，和の左端の桁は 1 である (図 10)．表記を統一するために、m 桁の虫食い算の左側に 2 列，右側に 1 列追加し，X_+ は $m+3$ 個の列，$C_{m+2}, C_{m+1}, C_m, \ldots, C_1, C_0$ から構成されるものとする (図 11)．具体的には，列 C_i の $1, 2, 3$ 行目をそれぞれ c_i^1, c_i^2, c_i^3 としたとき，$(c_0^1, c_0^2, c_0^3) := (0, 0, 0)$，$(c_m^1, c_m^2, c_m^3) := (0, 0, \alpha)$，$(c_{m+2}^1, c_{m+2}^2, c_{m+2}^3) := (0, 0, 0)$ とする．ここで，α は和が m 桁のとき 0，$(m+1)$ 桁のとき 1 とする．

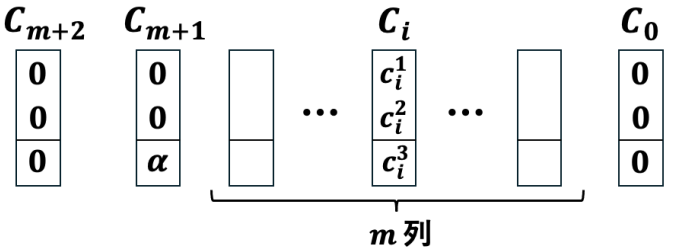

図 11: 定式化された虫食い算

次に，虫食い算 X_+ の解を定義する：

定義 3.2. 虫食い算 X_+ に含まれる変数を $x_1, \ldots, x_k$ とする．また，X_+ に含まれる変数への割り当てを $s \in \{t \in \mathbb{N} \mid 0 \leq t \leq n-1\}^k$ とし，各 $j \in \{t \in \mathbb{N} \mid 1 \leq t \leq k\}$ に対して，X_+ に含まれる x_j を s_j に置き換えて得られる筆算式を $X_+(s)$ とする．この $X_+(s)$ に矛盾が生じないとき，列 s は X_+ の解であるという．

筆算式の一貫性を確保するためには，個々の列だけでなく，隣接する列間の繰り上がりの値も考慮する必要がある．そのため，繰り上がりの値について，以下の表記法を導入する：

定義 3.3. $i = 0$ のとき：

$$\mathrm{Carry}(C_i(w)) := 0.$$

$1 \leq i \leq m+2$ のとき：

$$\mathrm{Carry}(C_i(w)) := \begin{cases} 0 & (c_i^1 + c_i^2 + c_i^3 + \mathrm{Carry}(C_{i-1}(w)) < n) \\ 1 & (c_i^1 + c_i^2 + c_i^3 + \mathrm{Carry}(C_{i-1}(w)) \geq n). \end{cases}$$

3.1.2 加算および減算の虫食い算の結果

まず，加算の虫食い算について次の定理が成立する：

定理 3.4. n 進数 m 桁の 2 数の加算に関する虫食い算が解を持つかどうかを $O(m \log n)$ 時間で判定することができる．また，解の構成および解の個数の数え上げも $O(m \log n)$ 時間で行うことができる．

また，減算の虫食い算についても，加算の場合と同様に次の定理が成り立つ：

定理 3.5. n 進数 m 桁の 2 数の減算に関する虫食い算が解を持つかどうかを $O(m \log n)$ 時間で判定することができる．また，解の構成および解の個数の数え上げも $O(m \log n)$ 時間で行うことができる．

定理 3.4 の証明は 2.1.4 節で，加算の虫食い算に関する $O(m \log n)$ 時間アルゴリズムを示すことで行う．また，定理 3.5 も同節で示す．

3.1.3 加算の虫食い算が解を持つ条件

加算の虫食い算が解を持つかどうかは，各列の繰り上がりの値の整合性にのみ依存する．具体的には，以下が成立する．

補題 3.6. 虫食い算 X_+ の i 列目を C_i とする．このとき，虫食い算 X_+ が解を持つことの必要十分条件は，ある長さ $m+1$ の 0 か 1 からなる列 $l \in \{0,1\}^{m+1}$ が存在し，「$1 \leq i \leq m+1$ を満たす任意の自然数 i に対して，$\mathrm{Carry}(C_i(w_i)) = l_i$ かつ $\mathrm{Carry}(C_{i-1}(w_{i-1})) = l_{i-1}$ を満たす，列 C_i に含まれる変数への割り当てが存在する」が成り立つことである．

証明. ($\Rightarrow$): 虫食い算 X_+ が解 s を持つとする．定義より，$X_+(s)$ は筆算として成立する．s による C_i への割り当てを w_i とし，$l = (\mathrm{Carry}(C_0(w_0)), \ldots, \mathrm{Carry}(C_m(w_m)))$ とすると，条件を満たすことは明らかである．

($\Leftarrow$) : 仮定より，各 i について，$\mathrm{Carry}(C_i(w_i)) = l_i$ かつ $\mathrm{Carry}(C_{i-1}(w_{i-1})) = l_{i-1}$ となる割り当て w_i が存在する．ここで，a_i^j を列 C_i の j 行目に，w_i の通りに割り当てられた数字もしくは，もとから虫食い算に入っていた数字とする ($i \in \{1,2,\ldots,m\}, j \in \{1,2,3\}$). 定義から，$a_i^1 \equiv a_i^2 + a_i^3 + \mathrm{Carry}(C_{i-1}(w_{i-1})) \pmod{n}$ が成立する．仮定より，$a_i^2 + a_i^3 + \mathrm{Carry}(C_{i-1}(w_{i-1})) < n$ が成立するのは $\mathrm{Carry}(C_i) = 0$ のときであり，$a_i^2 + a_i^3 + \mathrm{Carry}(C_{i-1}(w_{i-1})) \geq n$ が成立するのは $\mathrm{Carry}(C_i(w_i)) = 1$ のときである．従って，s を w_i に基づく全体への割り当てとすると $X_+(s)$ は筆算として成立する． □

上記の補題を用いてアルゴリズムを構築するには，各列の前後の繰り上がりの値が固定されたときの変数へ割り当てが存在するかどうかを判定する必要がある．これは以下の補題を用いて行うことができる：

補題 3.7. 虫食い算 X_+ のある列 C_i と $(\alpha, \beta) \in \{0,1\}^2$ に対して，C_i から C_{i+1} への繰り上がりの値が α かつ C_{i-1} から C_i への繰り上がりの値が β となる割り当て w が存在するか否かを $O(\log n)$ で判定することができる．

証明. 列 C_i を固定する ($1 \leq i \leq m$). また，行 $j \in \{1,2,3\}$ の変数または定数に対して，x^j, b^j, z^j を次で定義する：

$$(x^j, b^j, z^j) := \begin{cases} (c_i^j, 0, 1) & (c_i^j \text{が変数}) \\ (0, c_i^j, 0) & (c_i^j \text{が定数}) \end{cases}$$

ここで，列 C_i に対して以下が成り立つ．

$$\sum_{j=1}^{2}(x^j + b^j) + \beta = x^3 + b^3 + \alpha n$$

$$\Leftrightarrow \sum_{j=1}^{2} x^j - x^3 = b^3 - \sum_{j=1}^{2} b^j + \alpha n - \beta \tag{3.1}$$

$i < m$ の場合，式 (3.1) の左辺の値は $-z^3(n-1)$ から $(z^1+z^2)(n-1)$ までのすべての整数を取ることができる．よって，式 (3.1) の右辺の値が上記の範囲内のときかつそのときに限り，C_i への可能な割り当てが存在する．

$i = m$ のとき，c^3_{m+1} が 0 か 1 か，つまり X_+ の和が m 桁か $m+1$ 桁かの 2 つのケースについて考える．$c^3_{m+1} = 0$ の場合，c^1_m, c^2_m, c^3_m の変数に 0 を代入することはルールによって禁止されているため，式 (3.1) の右辺が $(z^1+z^2) - z^3(n-1)$ 以上 $(z^1+z^2)(n-1) - z^3$ 以下のときかつそのときに限り C_i の可能な割り当てが存在する．一方で $c^3_{m+1} = 1$ の場合，c^1_m, c^2_m の変数に 0 を代入することはルールによって禁止されているため，式 (3.1) の右辺が $(z^1+z^2) - z^3(n-1)$ 以上 $(z^1+z^2)(n-1)$ 以下のときかつそのときに限り C_i の可能な割り当てが存在する．

以上より，定数回の演算（四則演算，大小比較）によって C_i の可能な割り当てが存在することを判定することができる．また，各演算は $O(\log n)$ 時間で行うことができため，判定の全体にかかる計算時間も $O(\log n)$ である． □

ここで，補題 3.7 は可能な割り当ての存在を確認するだけでなく，可能な割り当ての例を生成し，そのような割り当ての数を数えるためにも使用することができる．これらは全て定数時間で実行可能なことに注意する．

3.1.4 加算の虫食い算の桁数に対する線形時間アルゴリズム

この節では虫食い算 X_+ を桁数に関する線形時間で解き，同時に解の個数を数え上げるアルゴリズムの概要とデータ構造について説明し，定理 3.4 の証明を行う．その後，X_+ の桁数に関する線形時間アルゴリズムを修正することで定理 3.5 の証明を行う．

まず，補題 3.6 より，与えられた虫食い算 X_+ の解を得るには，X_+ に含まれる C_i の前後の繰り上がりに矛盾が生じない繰り上がり方を見つければ良いことがわかる．つまり，ある長さ $m+1$ の 0 と 1 からなる列 $l \in \{0,1\}^{m+1}$ が存在して，各列 C_i の前後の繰り上がりが，それぞれ l_i，l_{i-1} となる変数の割り当てを見つければ良い．

ここで，ある列 C_i の前後の繰り上がり方 $(\mathrm{Carry}(C_i), \mathrm{Carry}(C_{i-1}))$ は $(0,0)$，$(0,1)$，$(1,0)$，$(1,1)$ の 4 パターンである．この 4 つのうちどのパターンが実現できるかを調べ，

2^m 通りの全ての繰り上がり方から各列ごとに割り当てが存在しない繰り上がりパターンを削除していくことで，可能な繰り上がりのパターンの集合を得ることができる．

次に，上記の手順をアルゴリズムとして定式化するために用いるグラフ構造について説明する．まず，図 12 のように X_+ の各列 C_i に対応する頂点 $v_{i,0}$，$v_{i,1}$ から構成されるグラフを用意する $(0 \leq i \leq m+2)$．

また，$v_{i,j}$ から $v_{i+1,k}$ に有効辺が張られているものとする $(j,k \in \{0,1\})$．

次に，以下の手順に従って，C_i に対応する 2 つの頂点それぞれから，C_{i+1} に対応する 2 つの頂点それぞれに向かって張られている 4 つの有効辺にラベルを付ける．ここで，$v_{i+1,j}$ に入る辺は列 C_i からの繰り上がりの値が j であることを表す．有効辺に対するラベルの付け方は以下である：

- $v_{i,0}$ から $v_{i+1,0}$ に張られた辺に，$(\mathrm{Carry}(C_{i+1}), \mathrm{Carry}(C_i)) = (0,0)$ となる割当の総数 $l^i_{(0,0)}$ をラベルとして付ける．
- $v_{i,1}$ から $v_{i+1,0}$ に張られた辺に，$(\mathrm{Carry}(C_{i+1}), \mathrm{Carry}(C_i)) = (0,1)$ となる割当の総数 $l^i_{(0,1)}$ をラベルとして付ける．
- $v_{i,0}$ から $v_{i+1,1}$ に張られた辺に，$(\mathrm{Carry}(C_{i+1}), \mathrm{Carry}(C_i)) = (1,0)$ となる割当の総数 $l^i_{(1,0)}$ をラベルとして付ける．
- $v_{i,1}$ から $v_{i+1,1}$ に張られた辺に，$(\mathrm{Carry}(C_{i+1}), \mathrm{Carry}(C_i)) = (1,1)$ となる割当の総数 $l^i_{(1,1)}$ をラベルとして付ける．

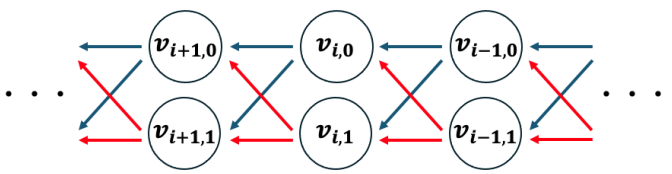

図 12: 可能な繰り上がりを表す構造を持つグラフ

虫食い算 X_+ の解の総数を求めるアルゴリズムの概要について説明する．まず，このアルゴリズムでは $i = m+1, m, m-1, \ldots, 0$ の順に，補題 3.7 を利用して，各列 C_i の変数に割り当てたとき実現できる前後の繰り上がり方と，その場合の数を求め，先述した方法でラベルを付ける．

このようにして構築したグラフ上で，列 C_0 に対応する頂点 $v_{0,0}, v_{0,1}$ をそれぞれ始点とした，列 C_{m+2} に対応する頂点 $v_{m+2,0}, v_{m+2,1}$ へのパスを考える．各パスについて含まれる辺に付けられたラベルを掛けながらたどり総和をとったものは，補題 3.6 から虫食い算 X_+ の解の総数である．

この解の総数を求める処理は，動的計画法を利用することによって m に関する線形時

間で実現できる．まず，頂点 $v_{0,0}, v_{0,1}$ のそれぞれから $v_{i,j}$ へのパスをたどりながらラベルの積を求め，総和をとったものを $\mathrm{dp}[i][j]$ とする．すると，$\mathrm{dp}[i+1][j]$ の値は以下の漸化式で表すことができる．ただし，$\mathrm{dp}[0][j] = 1$ とする．

$$\mathrm{dp}[i+1][j] = \begin{cases} l^i_{(0,0)} \cdot \mathrm{dp}[i][0] + l^i_{(0,1)} \cdot \mathrm{dp}[i][1] & (j = 0) \\ l^i_{(1,0)} \cdot \mathrm{dp}[i][0] + l^i_{(1,1)} \cdot \mathrm{dp}[i][1] & (j = 1) \end{cases}$$

よって，解の総数は $\mathrm{dp}[m+2][0] + \mathrm{dp}[m+2][1]$ である．

アルゴリズム全体の計算量について述べる．まず，X_+ に対応するグラフ上の頂点 $v_{i,j}$ から $v_{i+1,k}$ に張られた辺にラベルを付けるのは補題 3.7 を利用することによって，定数回の演算で実行可能であり，各演算（加算，大小比較）にかかる計算時間は $O(\log n)$ である．グラフ上の辺は全部で $4(m+1)$ 本であるため，すべての辺へのラベル付けは $O(m \log n)$ 時間で行える．また，この方法で構築したグラフを利用した解の総数の数え上げは，上述した動的計画法に基づく方法により $O(m)$ 時間で行うことができる．従って，X_+ の解の総数は $O(m \log n)$ 時間で求めることができる．

最後に，定理 3.5 の証明について，加算の虫食い算の m に関する線形時間アルゴリズムを修正することで，減算の虫食い算の m に関する線形時間アルゴリズムを得ることができる．具体的には，各列に含まれる変数に数字を割り当てたとき，可能な繰り下がりパターンを判定するように修正すれば良い．

3.1.5 k 数の加算の虫食い算について

本節では n 進数 m 桁の k 数の虫食い算 Z^k_+ の結果について述べる．ただし，Z^k_+ は 2 数の虫食い算の定義 3.1 と同様の方法で $0 \leq \alpha \leq k-1$ として定式化されており，$m+2$ 列で構成されるものとする．

補題 3.7 の系として以下が成立する．

補題 3.8. k 数の虫食い算 X^k_+ のある列 C_i と $(\alpha, \beta) \in \{t \in \mathbb{N} \mid 0 \leq t \leq k-1\}^2$ に対して，C_i から C_{i+1} への繰り上がりの値が α かつ C_{i-1} から C_i への繰り上がりの値が β となる割り当て w が存在するか否かを $O(k \log n)$ で判定することができる．

定理 3.4 を修正することで，k 数の加算の虫食い算について次が成立する：

定理 3.9. n 進数 m 桁の k 数の加算に関する虫食い算が解を持つかどうかを $O(k^3 m \log n)$ 時間で判定することができる．また，解の構成および解の個数の数え上げも $O(k^3 m \log n)$ 時間で行うことができる．

証明. 定理 3.4 の加算の虫食い算の桁数に関する線形時間アルゴリズムを修正することで，k 数の加算の虫食い算に対する m に関する線形時間アルゴリズムを得ることができ

る．具体的には，データ構造として k 数の加算の虫食い算 Z_+^k の各列 C_i に対応する頂点 $v_{i,0}, \ldots, v_{i,k-1}$ を含むグラフを用意して，各列に補題 3.8 を適用し，有効辺にラベルをつければ良い．また，解の個数の数え上げは定理 3.4 の証明中で用いている動的計画法による方法によって行うことができる．

計算時間については，グラフの辺の本数が $k^2(m+1)$ であり，各辺へのラベル付けは補題 3.8 により $O(k \log n)$ で行うことができるので，ラベル付けと解の総数の数え上げにかかる時間は $O(k^3 m \log n)$ である． □

3.2 除算の虫食い算

本節では，除算の虫食い算に関する結果を述べた後，除算の虫食い算の困難性の証明に利用する Matsui による乗算の虫食い算の結果 [10] について説明する．最後に，除算の虫食い算が NP 完全であることを証明する．

3.2.1 除算の虫食い算に関する結果

除算の虫食い算について次が成立する：

定理 3.10. n 進数 m 桁の 2 数の除算に関する虫食い算は NP 完全である．

この定理の証明は，2.2.2 節で，乗算の虫食い算からの帰着によって行う．

3.2.2 既存研究の手法と結果

2 数の乗算の虫食い算が NP 完全であることは Matsui によって示されており，NP 困難性の証明は Monotone 1-in-3 3SAT から還元することによって行われている [10]．ここで，Monotone 1-in-3 3SAT とは以下の問題である [13]：

入力：ちょうど 3 つの正の論理変数を含む節からなる CNF 論理式（3-CNF 論理式）．
問題：与えられた 3-CNF 論理式を充足させる，「各節に含まれる 3 つの論理変数のうち，ちょうど 1 つが真」となる論理変数への割り当てが存在するかどうかを判定する．

Matsui は証明の中で Monotone 1-in-3 3SAT に対応する乗算の虫食い算 $X_\times$ を構成した．この，$X_\times$ の概要を図 13 に示す．ここで，図 13 中の $S_A, S_B, S_C, S_1, S_2, \ldots, S_q$ は 1 桁の数字と空欄からなる列である．また，それぞれ $S_A, S_1, \ldots, S_q$ の長さは $1+pq(q+1)$ であり，S_B の長さは $p(q+1)(q-1)+1$，S_C の長さは $pq(q+1)+p(q+1)(q-1)+1$ である．

なお，本稿では $X_\times$ を元にして，これに対応する除算の虫食い算をどのように構成するのかに焦点を絞って考えるので，図 13 中の各行の構成方法などの細部についての説明は [10] を参照のこと．

3.2.3　定理 3.10 の証明

この問題が NP に属することは明らかである．次に，2 数の除算の虫食い算が NP 完全であることを，2 数の乗算の虫食い算から帰着することによって証明する．

まず，$X_\times$ に対応する除算の虫食い算 $X_\div$ を，$X_\times$ の構成要素 $S_A, S_B, S_C, S_1, S_2, \ldots, S_q$ と，$S_1, \ldots, S_q$ と同じ長さの空欄からなる列 $S'_1, \ldots, S'_q$ を図 14 のように組み合わせることで構成する．

ここで，$X_\div$ に含まれる最大の記号列 S_C の長さ $L(S_C)$ は多項式で表現できる．また，$X_\div$ は $q+3$ 個の記号列で構成されるので，$X_\div$ のサイズは高々 $(q+3)L(S_C)$ である．よって，与えられた Monotone 1-in-3 3SAT の入力に対応する $X_\div$ を多項式時間で構築することが可能である．

次に，構成した除算の虫食い算 $X_\div$ が NP 完全であることを示す．除算の筆算の途中式は，乗算の筆算の途中式の逆をたどったものであるから，この対応関係より，乗算の虫食い算 $X_\times$ を成立させる割り当ては除算の虫食い算 $X_\div$ を成立させることがわかる．なお，新しく追加した $S'_1, \ldots, S'_q$ については，他の行を埋めた後，減算を行うことによって割り当てを求めることができる．

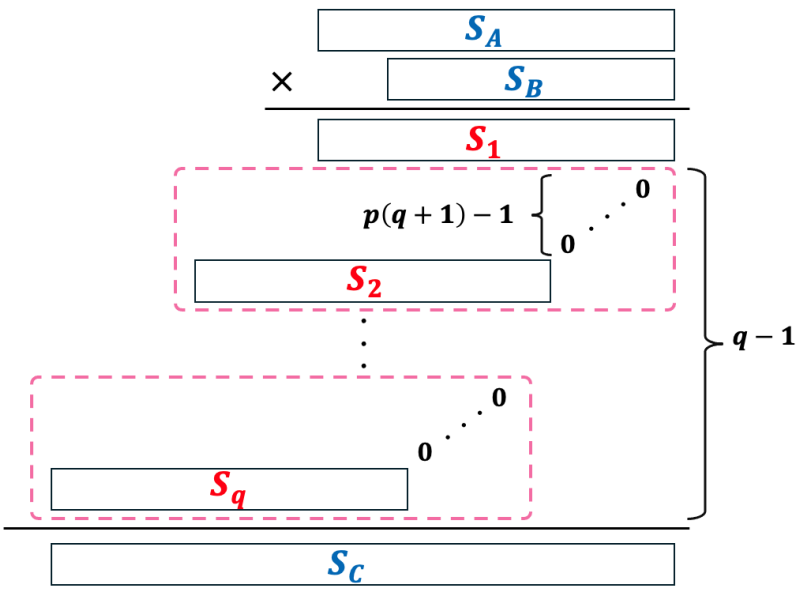

図 13: Monotone 1-in-3 3SAT に対応する乗算の虫食い算 $X_\times$ の概要

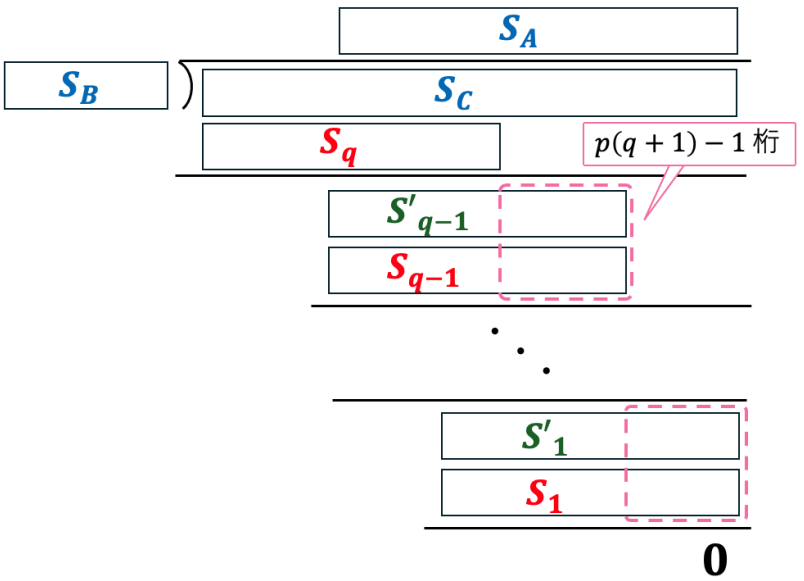

図 14: 乗算の虫食い算 $X_\times$ に対応する除算の虫食い算 $X_\div$

4　加算の 2 重虫食い算

本章では，「2 重虫食い算」と呼ばれる問題の計算複雑性について述べる．この問題は，2024 年の 7 月に池田洋介によって考案されたものである (図 15 の右側の問題)．

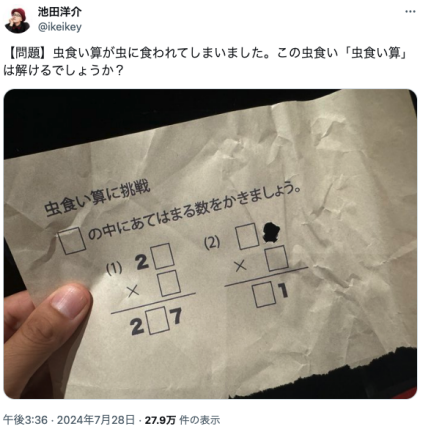

図 15: 池田氏が 2 重虫食い算を考案した際の投稿 [14]

4.1　加算の 2 重虫食い算の定義

2 重虫食い算では，図 16 のように所々が 2 重の虫食い (黒い空欄) に置き換えられた虫食い算が与えられる．ここで，図 16 の白い空欄は 1 重の虫食いであり，虫食い算に含まれる空欄のことである．この問題の目標は，2 の重虫食いに 1 重の虫食い (白い空欄)，一桁の数字のいずれかを割当てることで，唯一解を持つ虫食い算を完成させることである．ただし，最上位桁の 2 重虫食いに 0 を割り当てることは禁止されている．

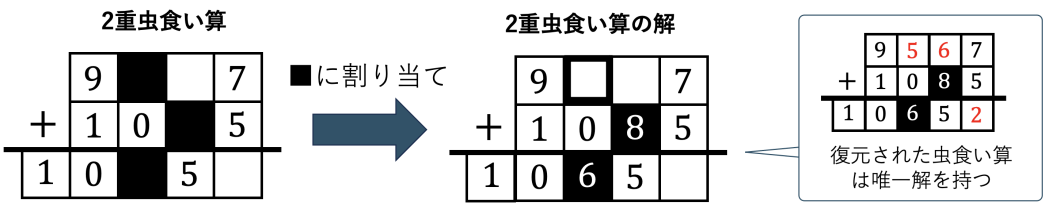

図 16: 加算の 2 重虫食い算の例とその解答

加算に関する 2 重虫食い算 Z_+ は定義 3.1，各列の繰り上がりについては定義 3.3 と同様の方法で定義されたものとする．以降，2 重虫食い算 Z_+ に含まれる，1 重の虫食いを

□，2 重の虫食いを■と書くことにする．次に，2 重虫食い算 Z_+ の解を定義する：

定義 4.1. Z_+ の列 C_i に含まれる■に対応する変数を $y_1, \ldots, y_{k_i}$ とする．また，$w \in \{t \in \mathbb{N} \mid 0 \leq t \leq n-1\}^{k_i}$ の列に対して，各 y_j を w_j に置き換えて得られる列を $C_i(w)$ とし，$y_1, \ldots, y_k$ を Z_+ 全体に含まれる■に対応する変数とする．ある数列 $s \in \{t \in \mathbb{N} \mid 0 \leq t \leq n-1\}^k$ に対して，各 y_j を s_j に置き換えて得られる虫食い算を $Z_+(s)$ とする．$Z_+(s)$ が唯一解を持つとき，列 s は Z_+ の解であるという．

4.2 加算の 2 重虫食い算のアルゴリズムとその証明

2 重虫食い算について次の定理が成立する．この定理の証明は，本節の最後に行う．

定理 4.2. n 進数 m 桁の 2 数の加算に関する 2 重虫食い算には $O(m \log n)$ アルゴリズムが存在する．

定理 4.2 のの証明では各列が■を含むかどうかによって場合分けを行う．そのため，Z_+ の各列 C_i について以下の関数を定義する．

定義 4.3. $0 \leq i \leq m+2$ に対して，次のように定義する．

$$\mathrm{Color}(C_i) := \begin{cases} W & (C_i\text{が■を含まない}) \\ B & (C_i\text{が■を少なくとも 1 つ含む}) \end{cases}$$

$\mathrm{Color}(C_i) = B$ を満たす列について，加算の虫食い算で $\mathrm{Color}(C_i) = W$ を満たす列の割り当てチェックに用いた補題 3.7 に対応する次の補題が成立する．この補題の証明は，4.3 節にて直接アルゴリズムを示すことによって行う．

補題 4.4. 2 重虫食い算 Z_+ の列 C_i で $\mathrm{Color}(C_i) = B$ を満たす列について，前後の繰り上がり $\mathrm{Carry}(C_i)$, $\mathrm{Carry}(C_{i-1})$ を固定した状態で，C_i に含まれる■に数字を割り当てたときに，□への割り当てが一意に定まるかどうかを $O(\log n)$ で確かめることができる．

上記で用意した補題 4.4 と，定理 3.4 の証明で利用した補題 3.7 を利用することによって，定理 4.2 を示す．具体的には，これらの補題を利用して繰り上がり方と解の個数を管理するグラフの辺にラベルを付けた後，唯一解に対応するパスに含まれない辺のラベルを 0 に書き換えていくことで証明を行う．

定理 4.2 の証明. 可能な繰り上がり方と割り当て方を管理するデータ構造として，Z_+ の各列 C_i に対応する頂点 $v_{i,0}, v_{i,1}$ から構成されるグラフを用意する $(0 \leq i \leq m+2)$．このグラフにおいて $v_{i,j}$ から $v_{i+1,k}$ に有効辺が張られているものとする $(j, k \in \{0, 1\})$．

具体的には，$i = m+1, m, m-1, \ldots, 0$ の順に，$\mathrm{Color}(C_i) = W$ のときは補題 3.7 を利用して，定理 3.4 の証明と同様の方法でラベル付けを行う．また，$\mathrm{Color}(C_i) = B$ のときは補題 4.4 を利用して，$(\mathrm{Carry}(C_{i+1}), \mathrm{Carry}(C_i)) = (\alpha, \beta)$ を固定した状態で■に数字を割り当てた際，□への割り当てが一意に定まるとき以下のようにラベル付けを行う．

- $(\alpha, \beta) = (0, 0)$ のとき，$v_{i,0}$ から $v_{i+1,0}$ に張られた辺にラベル 1 を付ける．
- $(\alpha, \beta) = (0, 1)$ のとき，$v_{i,1}$ から $v_{i+1,0}$ に張られた辺にラベル 1 を付ける．
- $(\alpha, \beta) = (1, 0)$ のとき，$v_{i,0}$ から $v_{i+1,1}$ に張られた辺にラベル 1 を付ける．
- $(\alpha, \beta) = (1, 1)$ のとき，$v_{i,1}$ から $v_{i+1,1}$ に張られた辺にラベル 1 を付ける．

いま，上記の方法でグラフのすべての辺にラベル付けが行われたとする．このとき，$i = m+1, m, \ldots, 0$ の順に列 C_i に対応する頂点をチェックしながら，唯一解に対応するパスに含まれない辺を除去していくことを考える．

列 C_i が $\mathrm{Color}(C_i) = W$ のときについて考える．$v_{i,j}$ から出る 2 本の辺のラベルが 0 でないとき，前後の繰り上がり方が $(0, j), (1, j)$ となる□への割り当てが存在するため，これらの辺は唯一解に対応するパスに含まれない．また，$v_{i,j}$ から出る辺のラベルが 2 以上のとき，□への割り当て方が 2 通り以上あるということなので，この辺も唯一解に対応するパスに含まれない．よって，これら 2 つのうち少なくとも一方を満たす辺を唯一解のチェックから除外するために，この辺のラベルを 0 に書き換える．

列 C_i が $\mathrm{Color}(C_i) = B$ のときについて考える．$v_{i,j}$ から出る辺のうち，ラベルが 1 の任意の辺は唯一解に対応するパスに含まれる．これは，補題 4.4 によって，この辺に対応する繰り上がり方のもとで C_i に含まれる□への割り当てが一意に定まることが保証されていることから明らかである．

上記の方法で得たグラフについて，$v_{0,0}$ から $v_{m+2,0}$ へのパスをたどりながらラベルの積を求めた値が 1 となるパスが存在したとする．このパスに対応するように，$\mathrm{Color}(C_i) = B$ の列に含まれる■に数字を割当てることによって，Z_+ に含まれる□への割り当てが一意に定まる．よって，補題 3.6 から Z_+ は唯一解を持つことがわかる．

最後に，計算時間について議論する．グラフの辺の本数が $4(m+1)$ であり，各辺へのラベル付けは補題 4.4 により $O(\log n)$ で行うことができるので，アルゴリズムの計算量は $O(m \log n)$ である．

□

4.3 補題 4.4 の証明

本節では補題 4.4 の証明を行う．具体的には，2 重虫食い■を含む列をパターンに分類し，各パターンごとに列の前後の繰り上がり方を固定したときに□への割り当てが一意になる■への割り当て方を求めることによって行う．

まず，2 重虫食い■を含む列の分類について説明する．列を構成するマスは，1 重の虫食い□・2 重の虫食い■・定数の 3 通りである．よって，■を含む列のパターン数は，全パターン 27 通りから■を含まないパターン 8 通りを引けば良いので 19 通りである．

ただし，図 17 のように同一の制約の列をまとめて考えると，各列のパターン数は 12 であることがわかる．また，図 17 に示すように各パターンに対して①–⑫と名前をつける．ただし，図 17 の x_j は未知変数，y_j は事前割り当て可能な変数，a_j は定数であり，それぞれ 2 重虫食い算の C_i 列目 j 行目の 1 重の虫食い，2 重の虫食い，プリセットの値に対応するものである ($j \in \{1,2,3\}$)．

2 重虫食い算ではリーディングゼロが禁止されているため，最上位に該当する列 C_m については別個に考える必要がある．

また，2 数の和の 2 重虫食い算において，和が m 桁のときに限り $\mathrm{Carry}(C_m) = 0$ となる．2 重虫食い算のルールより，列 C_m は 0 ではない定数を含み，この列 C_m に含まれる未知変数に 0 を割り当てることはできない．

一方で，虫食い算の和が $m+1$ 桁のときに限り $\mathrm{Carry}(C_m) = 1$ となる．このとき，2 重虫食い算のルールより 1,2 行目の値が 0 になることは禁止されている，つまり $x_k \neq 0$, $y_k \neq 0$, $a_k \neq 0$ である ($k \in \{1,2\}$)．また，3 行目の値についてはリーディングゼロ禁止の制約が課せられていないため，その限りではない．

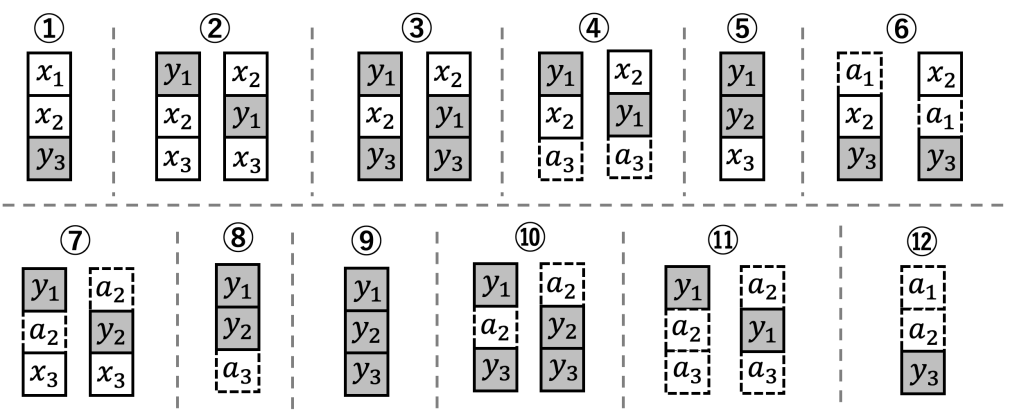

図 17: 2 重虫食い■を含む列のパターン

次に，2 重虫食い算を構成する列 C_i で，$\mathrm{Color}(C_i) = B_i$ を満たす列について，$1 \leq i \leq m-1$ の場合と $i = m$ の場合に分けて，「列の前後の繰り上がり方を固定した状態で■に数字を割り当てたときに，□の割り当てが一意になる」条件をパターンごとに示す．

パターン①：列 C_i が図のパターン①に該当するとき，以下が成立する．

$$x_1 + x_2 + \mathrm{Carry}(C_{i-1}) = y_3 + n \cdot \mathrm{Carry}(C_i).$$

このとき，列 C_i の前後の繰り上がり方を固定した状態で■に数字を割り当てたときに，□の割り当てが一意になる条件と，□への割り当てが一意になるような割り当て方の一例を示す．

$1 \leq i \leq m-1$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$x_1 = 0,\ x_2 = 0,\ y_3 = 0$
0	1	–	$x_1 = 0,\ x_2 = 0,\ y_3 = 1$
1	0	–	$x_1 = n-1,\ x_2 = n-1,\ y_3 = n-2$
1	1	–	$x_1 = n-1,\ x_2 = n-1,\ y_3 = n-1$

$i = m$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$x_1 = 1,\ x_2 = 1,\ y_3 = 2$
0	1	–	$x_1 = 1,\ x_2 = 1,\ y_3 = 3$
1	0	–	$x_1 = n-1,\ x_2 = n-1,\ y_3 = n-2$
1	1	–	$x_1 = n-1,\ x_2 = n-1,\ y_3 = n-1$

以降のパターンでは，煩雑のため同様の説明を省略し単に結果のみ示す．

パターン②：列 C_i が図のパターン②に該当するとき，以下が成立する．

$$y_1 + x_2 + \mathrm{Carry}(C_{i-1}) = x_3 + n \cdot \mathrm{Carry}(C_i).$$

$1 \leq i \leq m-1$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = n-1,\ x_2 = 0,\ x_3 = n-1$
0	1	–	$y_1 = n-2,\ x_2 = 0,\ x_3 = n-1$
1	0	–	$y_1 = 1,\ x_2 = n-1,\ x_3 = 0$
1	1	–	$y_1 = 0,\ x_2 = n-1,\ x_3 = 0$

$i = m$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = n-2,\ x_2 = 1,\ x_3 = n-1$
0	1	–	$y_1 = n-3,\ x_2 = 1,\ x_3 = n-1$
1	0	–	$y_1 = 1,\ x_2 = n-1,\ x_3 = 0$
1	1	–	–

パターン③：列 C_i が図のパターン③に該当するとき，以下が成立する．

$$y_1 + x_2 + \mathrm{Carry}(C_{i-1}) = y_3 + n \cdot \mathrm{Carry}(C_i).$$

$1 \le i \le m-1$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = 0,\ x_2 = 0,\ y_3 = 0$
0	1	–	$y_1 = 0,\ x_2 = 0,\ y_3 = 1$
1	0	–	$y_1 = n-1,\ x_2 = 1,\ y_3 = 0$
1	1	–	$y_1 = n-2,\ x_2 = 1,\ y_3 = 0$

$i = m$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = 1,\ x_2 = 1,\ y_3 = 2$
0	1	–	$y_1 = 1,\ x_2 = 1,\ y_3 = 3$
1	0	–	$y_1 = n-1,\ x_2 = 1,\ y_3 = 0$
1	1	–	$y_1 = n-2,\ x_2 = 1,\ y_3 = 0$

パターン④：列 C_i が図のパターン④に該当するとき，以下が成立する．

$$y_1 + x_2 + \mathrm{Carry}(C_{i-1}) = a_3 + n \cdot \mathrm{Carry}(C_i).$$

$1 \le i \le m-1$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = 0,\ x_2 = a_3$
0	1	$a_3 \ge 1$	$y_1 = 0,\ x_2 = a_3 - 1$
1	0	$a_3 \le n-2$	$y_1 = n-1,\ x_2 = a_3 + 1$
1	1	–	$y_1 = n-1,\ x_2 = a_3$

$i = m$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	$a_3 \geq 2$	$y_1 = 1,\ x_2 = a_3 - 1$
0	1	$a_3 \geq 3$	$y_1 = 1,\ x_2 = a_3 - 2$
1	0	$a_3 \leq n - 2$	$y_1 = n - 1,\ x_2 = a_3 + 1$
1	1	–	$y_1 = n - 1,\ x_2 = a_3$

パターン⑤：列 C_i が図のパターン⑤に該当するとき，以下が成立する．

$$y_1 + y_2 + \mathrm{Carry}(C_{i-1}) = x_3 + n \cdot \mathrm{Carry}(C_i).$$

$1 \leq i \leq m - 1$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = 0,\ y_2 = 0,\ x_3 = 0$
0	1	–	$y_1 = 0,\ y_2 = 0,\ x_3 = 1$
1	0	–	$y_1 = n - 1,\ y_2 = 1,\ x_3 = 0$
1	1	–	$y_1 = n - 2,\ y_2 = 1,\ x_3 = 0$

$i = m$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = 1,\ y_2 = 1,\ x_3 = 2$
0	1	–	$y_1 = 1,\ y_2 = 1,\ x_3 = 3$
1	0	–	$y_1 = n - 1,\ y_2 = 1,\ x_3 = 0$
1	1	–	$y_1 = n - 2,\ y_2 = 1,\ x_3 = 0$

パターン⑥：列 C_i が図のパターン⑥に該当するとき，以下が成立する．

$$a_1 + x_2 + \mathrm{Carry}(C_{i-1}) = y_3 + n \cdot \mathrm{Carry}(C_i).$$

$1 \leq i \leq m - 1$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$x_2 = 0,\ y_3 = a_1$
0	1	$a_1 \leq n - 2$	$x_2 = 0,\ y_3 = a_1 + 1$
1	0	–	$x_2 = n - a_1,\ y_3 = 0$
1	1	–	$x_2 = n - a_1 - 1,\ y_3 = 0$

$i = m$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	$a_1 \leq n-2$	$x_2 = 1,\ y_3 = a_1 + 1$
0	1	$a_1 \leq n-3$	$x_2 = 1,\ y_3 = a_1 + 2$
1	0	–	$x_2 = n - a_1,\ y_3 = 0$
1	1	–	$x_2 = n - a_1 - 1,\ y_3 = 0$

パターン⑦：列 C_i が図のパターン⑨に該当するとき，以下が成立する．

$$y_1 + a_2 + \mathrm{Carry}(C_{i-1}) = x_3 + n \cdot \mathrm{Carry}(C_i).$$

$1 \leq i \leq m-1$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	–	$y_1 = 0,\ x_3 = a_2$
0	1	$a_2 \leq n-2$	$y_1 = 0,\ x_3 = a_2 + 1$
1	0	$a_2 \geq 1$	$y_1 = n-1,\ x_3 = a_2 - 1$
1	1	–	$y_1 = n-1,\ x_3 = a_2$

$i = m$ のとき：

$\mathrm{Carry}(C_i)$	$\mathrm{Carry}(C_{i-1})$	定数の範囲	割当の一例
0	0	$a_2 \leq n-2$	$y_1 = 1,\ x_3 = a_2 + 1$
0	1	$a_2 \leq n-3$	$y_1 = 1,\ x_3 = a_2 + 2$
1	0	$a_2 \geq 1$	$y_1 = n-1,\ x_3 = a_2 - 1$
1	1	–	$y_1 = n-1,\ x_3 = a_2$

パターン⑧－⑫：

このとき，列に含まれる■を□とみなして補題 3.7 を利用することで，列の前後の繰り上がりパターンを満たす■への割り当ての存在を判定することができる．

以上より，各パターンでの判定は $O(\log n)$ 時間の演算（四則演算，大小比較）を定数回行うことで完了することがわかる．

5　まとめ

5.1　結果のまとめ

覆面算については：n 進数 m 桁の 2 数の乗算の覆面算が NP 完全であることを，n 進数 m 桁の 2 数の加算の覆面算からの還元によって証明した．次に，n 進数 m 桁の 2 数の減算の覆面算が NP 完全であることを，n 進数 m 桁の 2 数の加算の覆面算からの還元によって示した．その後，n 進数 m 桁の 2 数の除算の覆面算が NP 完全であることを，n 進数 m 桁の 2 数の減算の覆面算からの還元によって証明した．

虫食い算については：n 進数 m 桁の 2 数の加算・減算の虫食い算に $O(m \log n)$ の動的計画法に基づくアルゴリズムが存在することを示した．また，同様の方法で n 進数 m 桁の k 数の加算の虫食い算に $O(k^3 m \log n)$ に基づくアルゴリズムが存在することを証明した．

次に，n 進数 m 桁の 2 数の除算の虫食い算が NP 完全であることを，n 進数 m 桁の 2 数の乗算の虫食い算からの還元によって証明した．

最後に，虫食い算の拡張である，n 進数 m 桁の 2 数の加算の 2 重虫食い算に $O(m \log n)$ のアルゴリズムが存在することを証明した．

5.2　考察

乗算・除算の虫食い算，四則演算の覆面算は NP 完全だったが，加算・減算の虫食い算，加算の虫食い算を拡張した問題は NP 完全ではなかった．

前者が NP 完全となった理由についての考察を述べる．まず，パズルの構造から，乗算・除算の虫食い算に含まれる空欄や，覆面算に含まれる文字に数字を割り当てたとき，その空欄・文字を含む列以外に影響を及ぼす．各空欄・文字の影響がグローバルなため，各列のようなローカルなところではなく，複数の箇所に注目する必要があるため，他の NP 完全問題をシミュレートするような構造を埋め込むことができるレベルの複雑な問題となったと考えられる．

後者が NP 完全でない理由についての考察を述べる．まず，加算・減算の虫食い算に含まれる空欄に数字を割り当てたとき，その空欄を含む列の左隣のみに繰り上がりの有無として影響を及ぼす．各空欄の影響がローカルなため，各列ごとに繰り上がり方の条件を満たしているのかどうかのみをチェックすることで解の有無を判定することができるからである．

5.3 今後の展望

本研究では，n 進数 m 桁の 2 数の加算の 2 重虫食い算に $O(m \log n)$ のアルゴリズムが存在することを証明した．この 2 重虫食い算を，さらに 3 重，4 重と拡張していったときに，それぞれの問題の計算複雑性が階層構造をもつのかどうかというのは興味深い問題である．

また，n 進数 m 桁の 2 数の乗算の 2 重虫食い算の計算複雑性がどうなっているかというのも興味深い問題である．2 重虫食い算は，そのルールから虫食い算の FCP 版として扱える可能性がある．もし，FCP の枠組みで 2 重虫食い算を扱うことができれば，既に証明された NP 完全問題の FCP 版を利用することによって，乗算の 2 重虫食い算の計算複雑性を明らかにできる可能性がある．

謝辞

指導教員である上原先生には大変お世話になりました．急な研究室変更にも関わらず暖かく迎え入れて頂きありがとうございました．先生には，覆面算・虫食い算をテーマにしてはどうかと提案して頂き，その後も手厚く指導して頂きました．そのおかげで，なんとか 1 年でやりきれました．パズルの計算複雑性に関する研究は，私にとっては馴染が無くガジェットを作って他の問題をシミュレートするという手法に面食らってしまった思い出があります．その他にも勉強ゼミや研究室発表を通して，今までに馴染みのなかった知識に触れることができました．先生のおかげで今年は知的な刺激に溢れた年となりました，この場を借りて感謝申し上げます．また，鎌田先生にも本当にお世話になりました．いつ質問に行っても親身になって相談にのって頂いたり，何度も文章の添削をして頂いたりと本当にありがとうございました，先生のおかげスムーズに研究を進めることができ，充実した研究室生活を送れました．先生の存在なしでは修士論文の完成まで漕ぎ着けることはできなかったと思います，深く感謝致します．さらに，研究室の学生の皆様にも様々な場面でお世話になりました，厚く感謝致します．最後に，北陸での曇りがちな生活を明るく照らしてくれた友人たち，長い長い学生生活の中で私が勉学に励めるようにサポートし続けてくれた両親に心から感謝申し上げます．

参考文献

[1] 上原 隆平: パズルの算法 手とコンピュータでのパズルの味わい方, 日本評論社 (2024).

[2] Uehara, R.: *Computational Complexity of Puzzles and Related Topics*, Interdisciplinary Information Sciences, Vol. 29, No. 2, pp. 119–140 (2023).

[3] Uehara, R., Iwata, S.: *Generalized Hi-Q is NP-Complete*, Vol. E73–E, No. 2, pp. 270–273 (1990).

[4] Yato, T. and Seta, T.: *Complexity and Completeness of Finding Another Solution and Its Application to Puzzles*, IEICE Trans. Fundam. Electron. Commun. Comput. Sci., Vol. E86-A, No.5, pp.1052–1060 (2003).

[5] Kendall, G., Parkes, A., and Spoerer, K.: *A survey of NP-complete puzzles*, ICGA Journal, Vol. 31, No. 1, pp. 13–34 (2008).

[6] Seta, T.: *The complexities of puzzles, CROSS SUM and their another solution problems (ASP)*, Senior thesis, the University of Tokyo (2002).

[7] Hearn, R. A., and Demaine, E. D., *Games, Puzzles, and Computation*, A K Peters Ltd. (2009).

[8] Demaine, E. D., Ma, F., Schvarzman, A., Waingarten, E.: *The fewest clues problem*, Theoretical Computer Science, Vol. 748, pp. 28–39 (2018).

[9] Eppstein, D.: *On the NP-completeness of cryptarithms*, SIGACT NEWS, Vol. 18, No. 3, pp. 38–40 (1987).

[10] Matsui, T.: *NP-completeness of Arithmetical Restorations*, Journal of Information Processing, Vol. 21, No. 3, pp. 402–404 (2013).

[11] J.A.H. ハンター著, 藤村幸三郎・芹沢正三訳：数学ショートパズル, 付録 E, ダイヤモンド社 (1965).

[12] Tamori, S. and Akiyama, H.: Personal communication (2024).

[13] Garey, M.R. and Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman (1979).

[14] 池田 洋介: https://x.com/ikeikey/status/1817448938322202776 (2024).