信州大学
大学院理工学系研究科

修士論文

数独の消失通信路に対する復号誤り特性

指導教員　西新 幹彦 准教授

専攻　電気電子工学専攻
学籍番号　12TM244C
氏名　比田井 亮

2014 年 2 月 20 日

目次

1　はじめに

パズルの一つとして数独[*1]というものがある．数独は，かつてより親しまれてきたペンシルパズルの一種であり，空欄のマスに制約を満たす数字を書き込んでいくパズルである．このことから数独はもともとナンバープレースと呼ばれていた．ナンバープレースは 1970 年代から存在していたが，2005 年にイギリスで「Sudoku」の名で大流行したことをきっかけとして昨今では世界中で Sudoku の名で親しまれており，2006 年にはイタリアのルッカで第 1 回世界大会が開催され，2013 年には北京で開催された [1]．また，数独には一つのスタンダードなルールが存在するが，マスの数や数字の制約を変更することによってさまざまなバリエーションをいくらでも作ることができる．

数独はパズルとしてだけでなく，学問的な取り扱いも盛んに行われている．基本的な点としては，数独を解く問題は問題の大きさに関して NP 完全であることが知られている [2]．解の数に関する研究としては，数独の全ての解の数え上げ [3] とその番号付け [4] や，本質的に異なる解の数え上げ [5] とその番号付け [6]，解の数を確率的に推定する研究 [7] などがある．他に，物性物理学からのアプローチとして，一般化された数独の解の総数の平均場模型による評価 [8] や，数学からのアプローチとして，グレブナ基底を用いて問題を方程式として表現することによって代数的に数独を解く研究 [9] が行われている．最近の研究では数独の最小ヒント数が 17 であることが証明された [10]．

本研究では，パズルとして数独を解く過程を，通信における受信語から符号語を復号する過程とみなし，消失通信路に対する数独の復号誤りを調べることを目標とする．

2　数独のルール

本研究ではスタンダードなルールの数独を取り扱う．スタンダードな数独では，図 1[*2]にあるような 9 × 9 の 81 マスを用いる．81 個のマスは 9 個の 3 × 3 ブロックに分けられる．マスにはあらかじめ何ヶ所か数字が記入されており，プレーヤーは以下の制約を満たすようにすべての空欄に 1〜9 の数字のいずれかを入れていく．

- 一つのマスには 1 から 9 までのどれか一つの数字が入る
- どの行にも 1 から 9 までの数字が 1 回ずつ入る
- どの列にも 1 から 9 までの数字が 1 回ずつ入る
- 3 × 3 のブロックの中も 1 から 9 までの数字が 1 回ずつ入る

[*1] 「数独」は株式会社ニコリの登録商標である．

[*2] 「一般社団法人日本ナンプレ検定協会監修，難問 DX ナンプレ vol. 2，青空出版，2011 年」より改変．

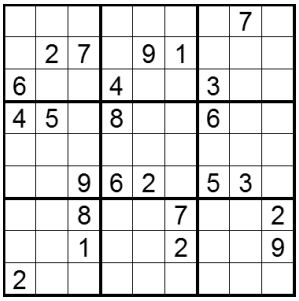

図 1　数独の例

すべてのマスに制約を満たした数字が入った状態を**解**という．あらかじめ記入されている数字はヒントとも呼ばれる．言うまでもなくヒントによって問題の難易度が決定される．ヒントの数が少なければ問題は難しくなる傾向にあるが，しかしその傾向は厳密ではない．ヒントの配置によっては複数の解が存在する場合があるが，そのような問題は「不良問題」と言われる．通常の数独の問題は解が唯一つ存在するようにヒントが配置されており，プレーヤーもそれを暗黙の仮定として解を推定する．

また，マス目が 9×9 ($= 3^2 \times 3^2$) の場合に限らず，一般にマス目が $n^2 \times n^2$ であれば問題を作ることができ，これを $n \times n$ 数独と呼ぶ．このときのルールは 3×3 数独と同様で，各列，各行，各ブロックに 1 から n^2 までの数字がそれぞれ 1 つずつ現れるように空欄に数字を入れればよい．実際に $4^2 \times 4^2$ ， $5^2 \times 5^2$ といったサイズの数独も目にすることがあるが，極めてまれである．本稿では特に断らない限り 3×3 数独を扱う．

ところで，数独の難易度は本質的には主観的なものである．しかし，客観的な尺度が無いわけではなく，マスに数字を入れるいくつかの定石が知られており，数字を入れられるマスを見つけ出すのにかかる手間は，定石によって明らかな違いがある．ちなみに，定石を使わなくても秩序立てて試行錯誤を繰り返せば必ず解にたどり着くことができるし，複数の解がある場合もすべての解を列挙することができる．

問題の難易度とは別に，愛好家は問題の面白さも重要視する．例えば，解にたどり着くためにどのような定石をどのような順番で適用するのか，ということが問題の面白さに影響する．「面白い問題」を自動生成するための方法がアミューズメントの分野では研究されている [11].

3　数独による通信システム

想定するシステムは次の通りである．数独の解を符号語として用いる．符号語は数独のすべての解の中から等確率に選ばれる．一つの符号語に対し，定常独立な消失通信路が 81 回使われて受信語が復号器に届く．つまり，1 マスに対して 1 回通信路がつかわれ，消失シンボルは

空白のマスとして受信される．受信語に対し，消失しなかった数字をヒントとして解が求められる．解が一意であれば誤りなく復号されたことを意味し，複数の解が存在すればそれらはそれぞれ等確率で正しい．

このシステムの基本的な性質を確認する．数独の解の総数 N_0 は

$$N_0 = 6670903752021072936960 \approx 6.67 \times 10^{21}$$

個であることが知られている [3] ので，符号語数は N_0 である．また，符号化率は $\frac{1}{81}\log_9 N_0 \approx 0.282$ である．消失確率 ϵ の消失通信路の通信路容量 C が $C = 1 - \epsilon$ であることから，数独と同じ符号化率の符号を用いたときに，任意に小さい復号誤りを達成できる消失確率の上限（シャノン限界）は $\epsilon \approx 0.718$ である．本研究の興味のひとつは，数独の復号誤り特性をシャノン限界と比較することである．

4　復号誤り特性の測定

本研究の目的は，消失通信路を数独によって符号化したときの平均復号誤り確率を求めることである．一般に，受信語に対して，消失シンボルの数と復号誤りの間には明確な関係はない．また，符号語によって復号誤り特性が異なる．しかし，だからといって N_0 個の解に対してあらゆる消失パターンを試すには膨大な時間がかかる．そのため，モンテカルロ法を用いて復号誤り特性を測定する．符号語は等確率で発生させる．

符号語を効率よく等確率で発生させることは簡単ではない．例えば，81 個のマスにランダムに数字を入れていき，解としての制約を満たさないものをあとから排除する愚直な方法を用いれば，解を等確率に選ぶことができる．しかし，この方法では解としての制約を満たすものが生成される確率は 3.39×10^{-56} であり，著しく効率が悪い．始めから制約を満たすものを選択することが重要である．

従来研究を組み合わせてこれを実現するには，1 から N_0 までの番号を等確率に発生させ，解の番号付け [4] にしたがって解を選び出せばよい．実はこの手続きを洗練させれば，全ての番号を網羅的に用意する必要なく，解を効率よく等確率で発生させることができる．これが本論文で提案するアルゴリズムである．またそのような理由から，提案アルゴリズムは解の総数を求めるアルゴリズムを内包している．つまり，誤り特性に関する部分を取り除くと解の総数を求めるアルゴリズムとなる．

なお，復号誤り特性は数独の問題としての難易度や解法のアルゴリズムとは関係がない．しかし，測定を効率よく行うために，効率のよい数独の解法が必要である．特に，試行錯誤は著しく効率が悪いので，知られている定石を用いることが必要となる．定石の詳細は本研究の本質ではないので，本論文では数独の解法アルゴリズムについては議論しない．

5　ランダムな解の生成

解を効率よく発生させるためには，解全体を階層的に適切な部分集合に分解することが重要である．部分集合のサイズを求めることによって，各部分集合が選ばれる確率が定まる．言い換えれば，解を葉とする木構造を導入することで，根から葉に至る経路をランダムに選べるようにするのが，提案法の基本的な考え方である．

説明のため，各マスには図 2 のように記号を付ける．$(s_{i1}, s_{i2}, \ldots, s_{i9})$ を第 i 行と呼び，$(s_{1j}, s_{2j}, \ldots, s_{9j})$ を第 j 列と呼ぶ．また，9 個の 3×3 ブロックにも図 3 のように記号を付ける．

5.1　解の絞り込み

数独の解全体を S_0 とおく．S_0 に属する解のうち，ブロック 1 が図 4 のようになる解を S_1 とおく．すると，S_1 に属する適当な解に対して，数字を置換することによって S_0 に属する任意の解を作ることができる．すなわち，S_0 を 9! 個の同サイズの部分集合に分解することが

s11	s12	s13	s14	s15	s16	s17	s18	s19
s21	s22	s23	s24	s25	s26	s27	s28	s29
s31	s32	s33	s34	s35	s36	s37	s38	s39
s41	s42	s43	s44	s45	s46	s47	s48	s49
s51	s52	s53	s54	s55	s56	s57	s58	s59
s61	s62	s63	s64	s65	s66	s67	s68	s69
s71	s72	s73	s74	s75	s76	s77	s78	s79
s81	s82	s83	s84	s85	s86	s87	s88	s89
s91	s92	s93	s94	s95	s96	s97	s98	s99

図 2　マスの番号付け

1	2	3
4	5	6
7	8	9

図 3　ブロックの番号付け

できる．また数字の置換によって復号誤りは変化しないことに注意すれば，S_1 に属する解を等確率に発生させればよいことが分かる．このように，誤り特性が等しい解の全単射が存在する 2 つの集合を**等価**であるという．

次に，S_1 に属する解に対して，$s_{14} < s_{15} < s_{16}$ となっている解全体を S_2 とおく．S_2 に属する解の第 4, 5, 6 列を入れ替えることによって S_1 に属する任意の解を作れることに注意しよう．したがって S_1 は S_2 とサイズの等しい 6 個の部分集合に分割できる．さらに S_2 の解のうち，第 7, 8, 9 列についても同様に $s_{17} < s_{18} < s_{19}$ となっている解全体を S_3 とおくと，S_2 は S_3 とサイズの等しい 6 個の部分集合に分割できることがわかる．ここで，S_3 の解を $s_{14} < s_{17}$ を満たすものとそうでないものが同数あることに注意して，満たすものを S_4 とおく．解の列を入れ替えても，定常独立消失通信路に対する誤り特性は変わらないことから，S_4 に属する解を等確率に発生させればよいことが分かる．

列の入れ替えと同様に，行の入れ替えを考えると，S_4 に属する解のうち，$s_{41} < s_{51} < s_{61}$, $s_{71} < s_{81} < s_{91}$, $s_{41} < s_{71}$ を満たす解全体を S_5 とおいて，S_5 に属する解を等確率に発生させればよいことが分かる．ちなみにここまでの議論で，$N_0 = |S_0| = 9! \times 72^2 \times |S_5|$ が成り立っている．また，S_5 に属する解を**標準形**と呼ぶ．

5.2 解の分類

ここまでで発生させる解を S_5 にまで絞り込んできた．ブロック 1,2,3,4,7 の値が等しいものによる S_5 の同値類分解を $\mathcal{S}_6$ とおく．本節では同値類分解 $\mathcal{S}_6$ を求める．$\mathcal{S}_6$ に属する各同値類は一般には等価ではない．しかし，次節以降のように注意深く調べれば，いくつかの同値類は等価であることがわかる．2 つの同値類が等価であればどちらか一方を調べることで他方の特性も分かる．従って，発生させる解をさらに絞り込むことができる．どのような同値類に分かれるのか，構造と数を確認する．

ブロック 1,2,3 の値が等しいものによる S_5 の同値類分解を $\mathcal{S}_7$ とおく．$\mathcal{S}_7$ に属する同値類

1	2	3
4	5	6
7	8	9

図 4　固定された第 1 ブロック

のうち，$(s_{14}, s_{15}, s_{16}, s_{17}, s_{18}, s_{19})$ が取りうる値は

$$(4,5,6,7,8,9) \tag{1}$$
$$(4,5,7,6,8,9) \tag{2}$$
$$(4,5,8,6,7,9) \tag{3}$$
$$(4,5,9,6,7,8) \tag{4}$$
$$(4,6,7,5,8,9) \tag{5}$$
$$(4,6,8,5,7,9) \tag{6}$$
$$(4,6,9,5,7,8) \tag{7}$$
$$(4,7,8,5,6,9) \tag{8}$$
$$(4,7,9,5,6,8) \tag{9}$$
$$(4,8,9,5,6,7) \tag{10}$$

のいずれかである．すなわち 10 種類の値を取りうるが，最初のひとつとそれ以外では性質が異なる．

まず，(1) の場合を考える．このとき，(s_{24}, s_{25}, s_{26}) は $\{7,8,9\}$ の順列である．これを $\{s_{24}, s_{25}, s_{26}\} = \{7,8,9\}$ と書く．同様に考えて結果をまとめると

$$\{s_{24}, s_{25}, s_{26}\} = \{7,8,9\}$$
$$\{s_{34}, s_{35}, s_{36}\} = \{1,2,3\}$$
$$\{s_{27}, s_{28}, s_{29}\} = \{1,2,3\}$$
$$\{s_{37}, s_{38}, s_{39}\} = \{7,8,9\}$$

となる (図 5)．これ以外の値は取ることができない．したがって $\mathcal{S}_7$ に属する同値類のうち $(s_{14}, s_{15}, s_{16}, s_{17}, s_{18}, s_{19}) = (4,5,6,7,8,9)$ となるものの数は $3! \times 3! \times 3! \times 3! = 1296$ 個である．

次に (2) の場合を考える．このとき他のマスの取りうる値は

$$\{s_{24}, s_{25}, s_{26}\} = \{8,9,a\}$$
$$\{s_{34}, s_{35}, s_{36}\} = \{6,b,c\}$$
$$\{s_{27}, s_{28}, s_{29}\} = \{7,b,c\}$$
$$\{s_{37}, s_{38}, s_{39}\} = \{4,5,a\}$$

1	2	3	4	5	6	7	8	9
4	5	6	{7,8,9}			{1,2,3}		
7	8	9	{1,2,3}			{4,5,6}		

図 5　上 1 行を固定したときの組み合わせ 1

1	2	3	4	5	7	6	8	9
4	5	6	{8,9,a}			{7,b,c}		
7	8	9	{6,b,c}			{4,5,a}		

図 6　上 1 行を固定したときの組み合わせ 2

となる（図 6）．ただし (a, b, c) は $(1, 2, 3)$, $(2, 3, 1)$, $(3, 1, 2)$ のいずれかである．したがって $\mathcal{S}_7$ に属する同値類
$(s_{14}, s_{15}, s_{16}, s_{17}, s_{18}, s_{19}) = (4, 5, 7, 6, 8, 9)$ となるものの数は $3 \times 3! \times 3! \times 3! \times 3! = 3888$ 個である．

(3) から (10) の場合も (2) の場合と同様で，それぞれの同値類は 3888 個存在する．したがって，$\mathcal{S}_7$ に属する同値類は 36288 個存在する．$\mathcal{S}_7$ と同様に，ブロック 1, 4, 7 の値による S_5 の同値類分解を $\mathcal{S}_8$ とおく．すると，$\mathcal{S}_8$ は $\mathcal{S}_7$ と同型であるため，$\mathcal{S}_8$ に属する同値類の数も 36288 個になる．さらに，$\mathcal{S}_6$ に属する任意の同値類 T は適当な $T_1 \in \mathcal{S}_7, T_2 \in \mathcal{S}_8$ を用いて $T = T_1 \cap T_2$ と書けるので，$\mathcal{S}_6$ に属する同値類の数は $36288 \times 36288 = 13146706944$ であることがわかる．これらの同値類の中には，互いに等価なものが存在する．それらを計算機によって予め探索することによって，S_5 のごく一部の解だけから復号誤りを測定することができる．

5.3　等価な同値類の探索

いま，改めて $T_1, T_2 \in \mathcal{S}_7$ を考え，T_1 と T_2 は等価だとする．5.4 節で説明するように，任意の $T \in \mathcal{S}_8$ に対して $T_1 \cap T$ と $T_2 \cap T$ は等価とは言えない．したがって，$\mathcal{S}_7, \mathcal{S}_8$ それぞれの中で等価な組を見つけても，$\mathcal{S}_6$ の中で等価な組を見つけたことにはならない．しかし，以下で述べるような「変換に基づく等価 [4][6]」という強い意味の等価関係を用いて $\mathcal{S}_7$ や $\mathcal{S}_8$ の構造を調べると，$\mathcal{S}_6$ の中で等価な組を見つけることができる．

5.3.1　行や列の入れ替えによる変換

どんな解に列や行の入れ替えをしてもその結果はやはりひとつの解であり，かつ復号誤りももとのものと等しい．この性質を用いてひとつの同値類 $S \in \mathcal{S}_7$ から別の等価な同値類に変換する手順がある．

行交換　集合 $S \in \mathcal{S}_7$ を考える．各 $t \in S$ に対して，第 1, 2, 3 行に置換を施した解を t' とおく（図 7）．すると，t' は S_1 の要素ではなくなる．しかし，マスの数字に関して適切な置換を施せば S_1 の要素となる．これを t'' とおく．すると，t'' は S_5 に属していないかも知れないが，第 4, 5, 6 列の入れ替えと第 7, 8, 9 列の入れ替え，そして必要ならばブロック 2, 5, 8 とブロック 3, 6, 9 をそれぞれ交換することによって s を作り，$s_{14} < s_{15} < s_{16}$, $s_{17} < s_{18} < s_{19}$,

$s_{14} < s_{17}$ とすることができる．行に関しても同様に入れ替えたものを s' とすれば $s' \in S_5$ となる．以降，解 t' から $s' \in S_5$ を導くこの操作を**標準化**と呼ぶことにする．さらに，s' は S と行の置換のみに依存して定まるある $S' \in \mathcal{S}_7$ に対して $s' \in S'$ となる．このとき S と S' は等価である．

列交換　集合 $S \in \mathcal{S}_7$ を考える．各 $t \in S$ に対して，第 1，2，3 列を入れ替えた解を t' とおく (図 8)．t' を標準化したものを s' とおくと，s' は S と第 1 ブロックの列の入れ替えのみに依存して定まるある $S' \in \mathcal{S}_7$ に対して $s' \in S'$ となる．このとき S と S' は等価である．

5.3.2　マスの入れ替えによる変換

解によっては，特定のマスの数字を入れ替えてもやはりひとつの解になる場合がある．この場合も復号誤りはもとのものと等しい．数字を入れ替えられる場所を見つけられればその変換により別の等価な同値類を得ることができる．

2 × 2 交換　集合 $S \in \mathcal{S}_7$ を考える．ある i_1, i_2, j_1, j_2 $(1 \le i_1, i_2 \le 3, 1 \le j_1, j_2 \le 9)$ が存在して，任意の $s \in S$ に対して $s_{i_1 j_1} = s_{i_2 j_2}$, $s_{i_1 j_2} = s_{i_2 j_1}$ だとする．このとき，$s_{i_1 j_1}$ と $s_{i_1 j_2}$，$s_{i_2 j_1}$ と $s_{i_2 j_2}$ をそれぞれ入れ替えたものもひとつの解となる（図 9）．この解を標準化したものを s' とおくと，ある $S' \in \mathcal{S}_7$ に対して，t によらず，$s' \in S'$ が成り立つ．このとき S と S' は等価である．

***k* × 2 交換**　2 × 2 交換を一般化し，k 個の列と 2 個の行を選び，選ばれたマスの各行が同じ数字からなっている場合も，標準化によって等価な同値類を見つけることができる．これを k × 2 交換と呼ぶ (図 10)．さらに 2 個の列と 3 個の行を選び，2 × 3 交換も考えることができる（図 11）．

***k* × 2 × 3 交換**　ブロック 1, 2, 3 から列をそれぞれ一つずつ選ぶ．各ブロックの選ばれた列をなす 3 つのマスから 2 つずつマスを選ぶ．ただし，ブロック 1, 2, 3 全体として，各行から 2 つのマスが選ばれるようにする．各行の選ばれたマスに入っている数字の組み合わせが 3 つとも同じであれば，各ブロック内で選ばれたマスの数字をそれぞれ同時に入れ替えてもそれはひとつの解となる．この解を標準化することによって等価な同値類を見つけることができる．こ

図 7　行交換の例

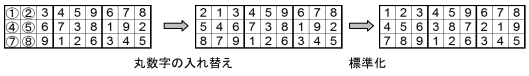

図 8　列交換の例

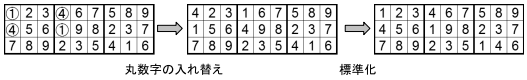

図 9　2 × 2 交換の例

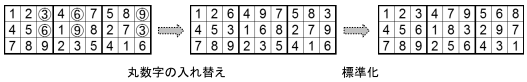

図 10　3 × 2 交換の例

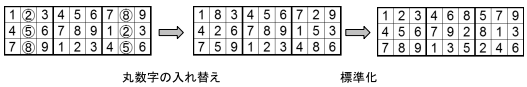

図 11　2 × 3 交換の例

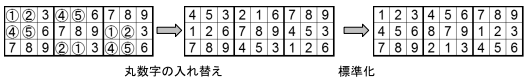

図 12　2 × 2 × 3 交換の例

れを 1 × 2 × 3 交換と呼ぶ．一般化によって $k \times 2 \times 3$ 交換を考えることもできる（図 12）．ところが実際のところ，1 × 2 × 3 交換は異なる同値類を作らないので意味がない．3 × 2 × 3 交換は [4] によって提案されている．

以上の交換を適用すると，$\mathcal{S}_7$ に属する同値類のうち，区別すべきなのは 209 種類となる．

5.4　ブロック 1, 4, 7 における解の絞込み

次にブロック 1, 4, 7 における同値類分解 $\mathcal{S}_8$ について考える．$\mathcal{S}_8$ は，$\mathcal{S}_7$ と同型ではあるが, これとは異なる方法で構造を調べなければならない．以下にその理由を説明する．

5.3 節で述べた各種変換はブロック 1,2,3 の変化に着目していたが, それに伴うブロック 4,7 の変化は無視している．すなわち，今までの変換は図 13 のようになっている．しかし，実際の交換では図 14 (a), (b) のように交換される．そのため，ブロック 1，4，7 に関して今まで行ってきた各種交換を行うと，ブロック 1，2，3 に関して等価かどうか考慮していないため, それぞれを交換したものを繋げた図 15 のような場合では等価な同値類となるか分からない．しかし，交換の中にはブロック 1，2，3 に関して考慮していなくてもブロック 1，2，3，4，7 における等価な同値類と分かるものも存在する．それらは，ブロック 1，4，7 において，ブロック 1 の数字については交換を行わず，ブロック 4，7 のみ交換を行うものである．一例を図 16 に示す．このようなの交換では，4，7 ブロック内の数字を入れ替えても 1，2，3 ブロッ

ク内の数字には変化がない．そのため，ブロック 1，2，3 に関して考慮していなくてもブロック 1，2，3，4，7 における等価な同値類となることが分かる．

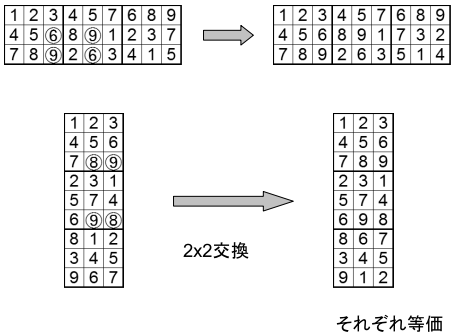

図 13　それぞれの交換

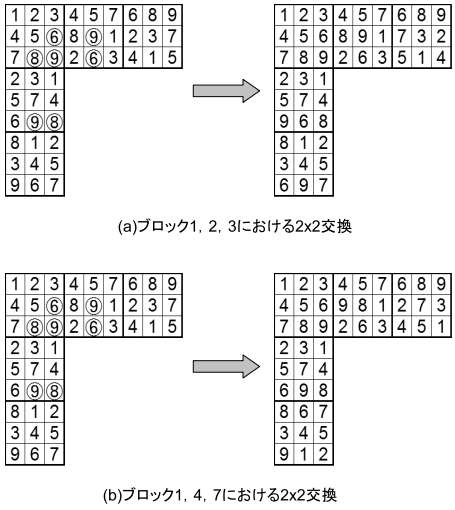

図 14　2 パターンの交換

従って，ブロック 1，4，7 における交換はブロック 1 におけるマスの置換を行わない変換を行うことで，解の関係を損なわずに探索範囲を少なくしている．ブロック 1，2，3 に依存しない交換は 5．3．2 節で議論した 2 × 2 交換と 2 × 3 交換において，ブロック 4，7 のみ数字の入れ替えが行われる場合である．この交換を用いるとブロック 1, 4, 7 における区別すべき同値類は 24408 個となる．各交換の可否を表 1 に示す．

したがって，数と誤り特性を調べるべき $\mathcal{S}_6$ に属する同値類は，$209 \times 24408 = 5101272$ 個で済むことになる．

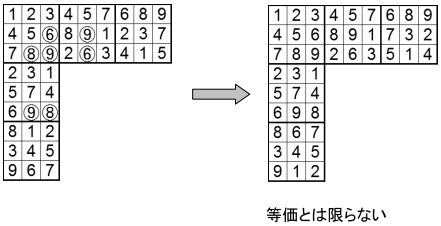

図 15　1，2，3，4，7 ブロックにおける交換

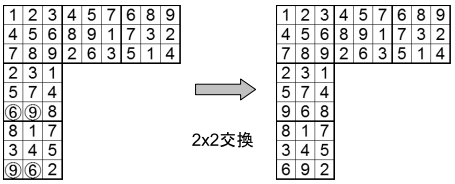

図 16　4，7 ブロックにおける交換例

表 1　交換の可否

ブロック	行変換	列変換	2 × 2	2 × 3	3 × 2	4 × 2	2 × 2 × 3	3 × 2 × 3
1，2，3	○	○	○	○	○	○	○	○
1，4，7	×	×	△	△	×	×	×	×

△…ブロック 4，7 のみ数字を入れ替え

従来研究 [4][6] を元に解の総数を求める手段を利用して等価な同値類をまとめた．しかし，解の総数を求める際には有効であったが本研究では利用していない変換もある．

5.5　解の重み付け

前節までは区別すべき同値類の個数について議論していたが，ここではその同値類をどの割合で発生させれば等確率に発生させられるか議論する．

まず，区別すべき $\mathcal{S}_6$ に属する同値類がいくつの同値類と等価であるのか調べ，番号をつける．その後，区別すべき同値類に対してそれぞれ解をいくつ持っているか数独を解くプログラムを用いて調べる．ブロック 1，2，3 における区別すべき同値類に対応する番号を i，ブロック 1，4，7 における区別すべき同値類に対応する番号を j とする．区別すべき同値類における変換によってまとめられた等価な自身を含む同値類の数を $X_{(i,j)}$，区別すべき同値類に含まれている解の個数を $Y_{(i,j)}$ とする．ここで i と j の範囲はそれぞれ $1 \leqq i \leqq 209$，$1 \leqq j \leqq 24408$ である．このとき各番号における区別すべき同値類の重みを次のように定義する．

$$X_{(i,j)} \times Y_{(i,j)}$$

同値類の重みとは同値類に含まれている解の総数のことなので，すべての解を一様に発生させたことにするには，重みが大きい同値類は確率を大きく，重みが小さい同値類は確率を小さくすればよい．ブロック 1，2，3，4，7 における区別すべき解を $Z_{(i,j)}$ とするとその発生確率 $P(Z_{(i,j)})$ は区別すべき同値類における番号 (i,j) の同値類が選ばれる確率は $\frac{X_{(i,j)} \times Y_{(i,j)}}{|S_5|}$，選ばれた同値類に含まれている解の中から 1 つ解を選ぶ確率は $\frac{1}{Y_{(i,j)}}$ である事から

$$P(Z_{(i,j)}) = \frac{X_{(i,j)} \times Y_{(i,j)}}{|S_5|} \times \frac{1}{Y_{(i,j)}}$$

となる．

以上の計算のため，$X_{(i,j)}, Y_{(i,j)}$ の値を調べた．各番号の重みの総和を取ることで S_5 のサイズを求めることができ，その値は次のようになった．

$$|S_5| = \sum_i \sum_j X_{(i,j)} \times Y_{(i,j)} = 3546146300288$$

これを $9! \times 72^2 \times |S_5|$ に代入したところ，解の総数 N_0 と一致することが確認できた．

6 復号誤り特性

受信語の空白のマスの数が 3 以下のときは解は一意に決まるので，復号誤りは起きないが，4 以上では不良問題となる場合がある．また，文献 [10] より，空白のマスの数が 65 以上では必ず不良問題となる．本研究では，計測の簡単のため，不良問題は誤りとみなす．よって測定すべきなのは空白のマスの個数が 4 から 64 の範囲である．具体的には，空白のマスは，個数を 4 から 64 の間で指定した後，場所に関してはランダムに場所を指定し，指定した場所を空白にする．空白が k 個の時の平均復号誤り確率 f_k を測定し，次のように計算を行うことで復号誤り特性を求めた．

消失確率が ϵ のとき k 個のマスが空白になる確率 $p_k(\epsilon)$ は

$$p_k(\epsilon) = \binom{81}{k} (\epsilon)^k \times (1-\epsilon)^{81-k} \tag{1}$$

復号誤り特性 $F(\epsilon)$ は

$$F(\epsilon) = \sum_{k=0}^{81} p_k(\epsilon) \times f_k \tag{2}$$

となる．

結果を図 17 に示す．縦軸は平均復号誤り確率，横軸は消失しない確率となっており，グラフ中にある縦線は各数独におけるシャノン限界を示している．

解が複数出てきた際，

$$\frac{1}{\text{出てきた解の総数}}$$

の確率で正しい解が発生するとすればより正確な特性が得られると考えられる．しかし，このような方法を採る場合，とりうる全ての解を発生させなければならない．その場合，空白が多くなると解の総数を調べるのに非常に時間がかかるため，このような方法は採用しなかった．

また，規模を 16 マスに縮小した 2 × 2 数独における復号誤り特性についても調べたのでその結果を図 17 に併せて記載する．詳しい手順は付録にまとめる．

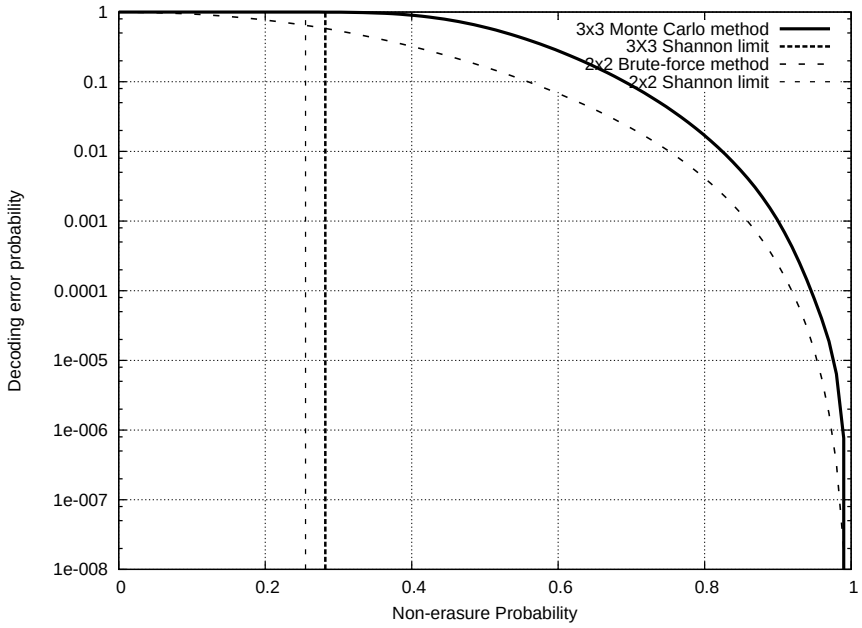

図 17　数独における復号誤り特性

7　まとめ

数独は，かつてより親しまれてきたペンシルパズルの一種であり，空欄のマスに制約を満たす数字を書き込んでいくパズルである．広く親しまれている 3 × 3 数独に注目し，パズルとして数独を解く過程を，通信における受信語から符号語を復号する過程とみなし，消失通信路に対する数独の復号誤り特性を計測した．測定の際，複数の解が発生した場合には誤りとみなし

ていたが，複数の解が存在すればそれらはそれぞれ等確率で正しいとすれば，より正確な特性が得られると考えられる．しかし，その際には空白が多いほど時間もかかるため，本研究ではこのような測定を行わなかった．

また，規模を縮小した 2 × 2 数独に関しても消失通信路に対する復号誤り特性を求めた．こちらは複数の解が存在すればそれらはそれぞれ等確率で正しいとして測定を行い，正確な誤り特性を得た．

謝辞

本研究に当たり，細かく指導してくださった指導教員の西新幹彦准教授に感謝の意を表する．

参考文献

[1] World puzzle Federation, `http://www.worldpuzzle.org/`, 2013 年 12 月閲覧.

[2] T. Yato, T. Seta, “Complexity and completeness of finding another solution and its application to puzzles,” IEICE trans. fundamentals, pp.1052-1060, 2003.

[3] Bertram Felgenhauer, Frazer Jarvis, “Enumerating possible Sudoku grids,” `http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf`, 2013 年 12 月閲覧.

[4] 戸神星也，「数独の解生成と解に対する番号付け」，東京工業大学理学部情報科学科卒業論文， 2006.

[5] Ed Russell, Frazer Jarvis, “Sudoku enumeration,” `http://www.afjarvis.staff.shef.ac.uk/sudoku/sudgroup.html`, 2013 年 12 月閲覧.

[6] 井上真大，奥乃博，「本質的に異なる数独解盤面の列挙と番号付け」，情報処理学会，第 71 回全国大会講演論文集，vol.4，pp．741–742，2009.

[7] 貞廣泰造，「数独解盤面の近似数え上げ」，数学セミナー, vol.51, no.3，pp．26–29，2012.

[8] 田中利幸，福島孝治，「数独に対する平均場模型」，日本物理学会講演概要集，pp．264，2010.

[9] 井上秀太郎，佐藤洋介，鈴木晃，鍋島克輔，「ブーリアングレブナ基底を使った数独の解法」，数理解析研究所講究録第 1666 巻，pp．1–5，2009.

[10] Gary McGuire，Bastian Tugemann，GillesCivario，“There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem,” School of Mathematical Sciences, University College Dublin, Ireland，2013.

[11] 前田一貴，奥乃博，「数独の問題作成支援システムの設計と開発」，情報処理学会，第 70 回全国大会講演論文集，vol.4，pp.799–800，2008.

付録 A　2×2 数独による通信システム

ここでは消失通信路を 2×2 数独によって符号化した際の性質と測定方法について説明する．2×2 数独の解の例を図 18 に示す.

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

図 18　2×2 数独の解の例

想定するシステムは次の通りである．数独の解を符号語として用いる．符号語は数独のすべての解の中から等確率に選ばれる．一つの符号語に対し，定常独立な消失通信路が 16 回使われて受信語が復号器に届く．つまり，1 マスに対して 1 回通信路が使われ，消失シンボルは空白のマスとして受信される．受信語に対し，消失しなかった数字をヒントとして解が求められる．解が一意であれば誤りなく復号されたことを意味し，複数の解が存在すればそれらはそれぞれ等確率で正しい.

このシステムの基本的な性質を確認する．2×2 数独の解の総数は 288 個である．また，符号化率は $\frac{1}{16} \log_4 288 \approx 0.255$ である．消失確率 ϵ の消失通信路の通信路容量 C が $C = 1 - \epsilon$ であることから，数独と同じ符号化率の符号を用いたときに，任意に小さい復号誤りを達成できる消失確率の上限（シャノン限界）は $\epsilon \approx 0.745$ である.

測定方法は，2×2 数独の全ての解に対し，空白のできるパターンを全て作成し，ルールに基づいて数独を解くことで各解，各空白パターンにおける受信側での取りうる解の総数 N が求められる．受信側において，複数個の解が出てきた時，そのうちの 1 つが受信語となる．従って，正しく送信できる確率は $\dfrac{1}{N}$ である．以上のようにして空白の数ごとに平均復号誤り確率を求め，式 (1)，式 (2) と同様の計算を行うことで消失確率に対する 2×2 数独の復号誤り特性を求めた（図 17）.

付録 B　ソースコード

以下は復号誤り特性を測定する際に用いたプログラムである.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#define MRND 1000000000L
static int jrand;
static long ia[56];

#define rmax 4106

int r_permu[9] = {'1', '4', '7', '2', '5', '8', '3', '6', '9'};
int rule[27][9];
short x_c[81];

/*K 乱数はじめ*/
static void irn55(void)
{
    int i;
    long j;
    for (i = 1;i <= 24; i++){
        j = ia[i] - ia[i+31];
        if (j < 0) j += MRND;
        ia[i] = j;
    }
    for (i = 25; i <= 55; i++){
        j=ia[i] - ia[i-24];
        if (j < 0) j += MRND;
        ia[i] = j;
    }
}
void init_rnd(unsigned long seed)
{
    int i,ii;
    long k;
    ia[55] = seed;
    k = 1;
    for (i = 1; i <= 54; i++){
        ii = (21 * i) % 55;
        ia[ii] = k;
        k = seed - k;
        if (k < 0) k += MRND;
        seed = ia[ii];
    }
    irn55(); irn55(); irn55();
    jrand = 55;
}
long irnd(void)
{
    if (++jrand > 55) {irn55(); jrand=1;}
    return ia[jrand];
}
double rnd(void)
{
    return (1.0/MRND) * irnd();
}
/*K 乱数終わり*/

void
charcpy(char *dist, char *str)
{
    while (*str != '\0'){
```

```
        *dist = *str;
        dist++;
        str++;
    }
    return;
}

void
charcpy_p(char *dist, char *str, int *permu)
{
    int i;

    for (i = 0; str[i] != '\0'; i++){
        *dist = str[permu[i]];
        dist++;
    }
    return;
}

void
charcpy9(char *dist, char *str)
{
    int i;

    for (i = 0; i < 9; i++){
        *dist++ = *str++;
    }
    return;
}

void
print_sudoku(short c[])
{
    int i, n;

    for (i = 0; i < 81; i++){
        switch (c[i]){
        case 0x000: n = 'X'; break;
        case 0x001: n = '1'; break;
        case 0x002: n = '2'; break;
        case 0x004: n = '3'; break;
        case 0x008: n = '4'; break;
        case 0x010: n = '5'; break;
        case 0x020: n = '6'; break;
        case 0x040: n = '7'; break;
        case 0x080: n = '8'; break;
        case 0x100: n = '9'; break;
        default:    n = ' '; break;
        }
        putchar(n);
        if (i % 9 == 8){
            putchar('\n');
        }
    }
    putchar('\n');
    return;
}


void
set_rule()
{
    int i, j;

    for (i = 0; i < 9; i++){
        for (j = 0; j < 9; j++){
            rule[i][j] = i * 9 + j;
            rule[i + 9][j] = i + j * 9;
            rule[i + 18][j] = (i / 3) * 18 + i * 3 + (j / 3) * 6 + j;
```

```
        }
    }
    return;
}

int permu3[6][3] = {
    {0, 1, 2}, {0, 2, 1},
    {1, 0, 2}, {1, 2, 0},
    {2, 0, 1}, {2, 1, 0}};

void
dec(char lines[81], int m)
/*上 3 ブロックに対して番号に対応した標準形を作成する*/
{
    int a0, a1, a2, a3, a4, a5;
    int i;

    a0 = m % 6; m /= 6;
    a1 = m % 6; m /= 6;
    a2 = m % 6; m /= 6;
    a3 = m % 6; m /= 6;
    a4 = m % 3; m /= 3;
    a5 = m;

    for (i = 0; i < 81; i++){
        lines[i] = ' ';
    }

    if (a5 == 9){
        charcpy(lines, "123456789456");
        charcpy_p(lines + 12, "789", permu3[a3]);
        charcpy_p(lines + 15, "123", permu3[a2]);
        charcpy(  lines + 18, "789");
        charcpy_p(lines + 21, "123", permu3[a1]);
        charcpy_p(lines + 24, "456", permu3[a0]);
    } else {
        static char *firstline[9] = {
            "457689", "458679", "459678", "467589", "468579",
            "469578", "478569", "479568", "489567" };
        static char tri1[9][4] = {"89a", "79a", "78a", "89a", "79a", "78a", "9bc", "8bc", "7bc" };
        static char tri2[9][4] = {"7bc", "8bc", "9bc", "7bc", "8bc", "9bc", "78a", "79a", "89a" };
        static char tri3[9][4] = {"6bc", "6bc", "6bc", "5bc", "5bc", "5bc", "56a", "56a", "56a" };
        static char tri4[9][4] = {"45a", "45a", "45a", "46a", "46a", "46a", "4bc", "4bc", "4bc" };
        static char *cana[3] = {"123", "231", "312"};
        static int idxa[9] = {0, 0, 0, 0, 0, 0, 1, 1, 1};

        if (idxa[a5] == 0){
            tri1[a5][2] = cana[a4][0];
            tri2[a5][1] = cana[a4][1];
            tri2[a5][2] = cana[a4][2];
            tri3[a5][1] = cana[a4][1];
            tri3[a5][2] = cana[a4][2];
            tri4[a5][2] = cana[a4][0];
        } else {
            tri1[a5][1] = cana[a4][1];
            tri1[a5][2] = cana[a4][2];
            tri2[a5][2] = cana[a4][0];
            tri3[a5][2] = cana[a4][0];
            tri4[a5][1] = cana[a4][1];
            tri4[a5][2] = cana[a4][2];
        }

        charcpy(  lines +  0, "123");
        charcpy(  lines +  3, firstline[a5]);
        charcpy(  lines +  9, "456");
        charcpy_p(lines + 12, tri1[a5], permu3[a3]);
        charcpy_p(lines + 15, tri2[a5], permu3[a2]);
        charcpy(  lines + 18, "789");
        charcpy_p(lines + 21, tri3[a5], permu3[a1]);
```

```
        charcpy_p(lines + 24, tri4[a5], permu3[a0]);
    }
    return;
}

void
r_dec(char lines[81], int n)
/*左縦 3 ブロックに対して番号に対応した標準形を作る*/
{

    char modlines[81];
    int i, j;
    dec(modlines,n);
    for (i = 0; i < 27; i++){
        modlines[i] = r_permu[modlines[i] - '1'];
    }
    for (j = 0; j < 6; j++){
        lines[27 + 9 * j] = modlines[3 + j];
        lines[28 + 9 * j] = modlines[12 + j];
        lines[29 + 9 * j] = modlines[21 + j];
    }

    return;
}

int
bit_count(int x)
{
    int x1 = (x  & 0x555) + ((x  & 0x0aa) >> 1);
    int x2 = (x1 & 0x333) + ((x1 & 0x0cc) >> 2);
    int x3 = (x2 & 0xf0f) + ((x2 & 0x0f0) >> 4);
    int x4 = (x3 & 0x0ff) + ((x3 & 0xf00) >> 8);
    return(x4);
}

int
deduce1(short c[])
/* 数字が確定している場所があればその数字は他の場所にはない */
{
    int i, j, k, m;

    m = 0;
    for (i = 0; i < 27; i++){
        for (j = 0; j < 9; j++){
            if (bit_count(c[rule[i][j]]) == 1){
                int k;
                for (k = 0; k < 9; k++){
                    if (k != j){
                        int b = c[rule[i][k]] & ~c[rule[i][j]];
                        if (c[rule[i][k]] != b){
                            c[rule[i][k]] = b;
                            m = 1; /* 更新 */
                        }
                    }
                }
            }
        }
    }
    return(m);
}

int
deduce2(short c[])
    /* 場所が確定している数字があればその場所には他の数字はない */
{
    int i, j, k, m;

    m = 0;
    for (i = 0; i < 27; i++){
```

```
        for (j = 0; j < 9; j++){
            int bit = 1 << j;
            int index;
            int dup = 0;
            for (k = 0; k < 9; k++){
                if (c[rule[i][k]] & bit){
                    index = k;
                    dup++;
                }
            }
            if (dup == 1 && c[rule[i][index]] != bit){
                c[rule[i][index]] = bit;
                m = 1; /* 更新 */
            }
        }
    }
    return(m);
}

unsigned long
search_sol(short c[], int n)
    /* 仮置きして解を全探索 */
{
    unsigned long num_sol;
    short can[81];
    int s_num, s_idx;
    int i, j, k, l;
    for (;;){
        if (deduce1(c)){
            continue;
        }
        if (deduce2(c)){
            continue;
        }
        break;
    }

    s_num = 10;
    for (i = 0; i < 81;i++){
        int b_c;
        if (c[i] == 0x0000){
            return(0); /* 解なし */
        }
        b_c = bit_count(c[i]);
        if (b_c > 1 && b_c < s_num){
            s_num = b_c;
            s_idx = i;
        }
    }
    if (s_num == 10){
        for(i = 0; i < 81; i++){
            x_c[i] = c[i];
        }
        return(1); /* 解である */
    }

    l = 0;
    num_sol = 0;
    for (j = 0; j < 9; j++){
        if (!((1 << j) & c[s_idx])){
            continue;
        }
        for (i = 0; i < 81; i++){
            can[i] = c[i];
        }
        can[s_idx] = 1 << j;
        num_sol += search_sol(can, n);
        if(num_sol >= n && n != 0){
            break;
```

```
        }
    }
    return(num_sol);
}

struct a{
    int left;
    double right;
};

struct b{
    int left2;
    double right2;
};

void
make(char c[], short s_c[])
/*盤面作成*/
{
    FILE *fp = fopen("c_p.txt", "r");
    if(fp == NULL){
        printf("error in %d\n", __LINE__);
        exit(1);
    }

    int i = 0;
    int j = 0;
    int k;
    int n, m;
    double x1, x2, x3;
    int num_sol;
    int left, left2;
    double right, right2;
    char filename[10];
    struct a ar[209];
    struct b br[4200];

/*上３読み込み*/
    while(feof(fp) == 0){
        fscanf(fp, "%d\t%lf\n", &left, &right);

        ar[i].left = left;
        ar[i].right = right;
        i++;
    }
    fclose(fp);
    x1 = rnd();
    for(i = 0; i < 209; i++){
        if(x1 <= ar[i].right){
            n = i;
            break;
        }
    }
    sprintf(filename, "%d.txt", ar[n].left);
    dec(c, ar[n].left);
/*左３読み込み*/
    FILE *fp2 =fopen(filename, "r");
    if(fp2 == NULL){
        printf("error in %d\n", __LINE__);
        exit(2);
    }
    while(feof(fp2) == 0){
        fscanf(fp2, "%d\t%lf\n", &left2, &right2);

        br[i].left2 = left2;
        br[i].right2 = right2;

        i++;
    }
```

```
    fclose(fp2);
    x2 = rnd();
    for(j = 0; j < 4200; j++){
        if(x1 <= br[j].right2){
            m = j;
            break;
        }
    }
    r_dec(c, br[m].left2);

    for (j = 0; j < 81; j++){
        if (c[j] == ' '){
            s_c[j] = 0x1ff;
        } else {
            s_c[j] = 0x1 << (c[j] - '1');
        }
    }

/*残りのマス埋め*/
    num_sol = search_sol(s_c, 0);
    //printf("num_sol = %d\n", num_sol);
    x3 = rnd();
    for(k = 1; k < num_sol; k++){
        if(x3 <= ((double)k / (double)num_sol)){
            break;
        }
    }
    search_sol(s_c, k);
    return;
}

void
shuffle(int n, int v[])
{
    int i, j, t;

    for(i = n - 1; i > 0; i--){
        j = (int)((i + 1) * rnd());
        t = v[i];
        v[i] = v[j];
        v[j] = t;
    }
}

void
randperm(int n, int v[])
{
    int i;
    for(i = 0; i < n; i++){
        v[i] = i;
    }
    shuffle(n, v);
}
void
boring(short c[], int n)
/*穴あけ*/
{
    int i, j, k;
    int v[81];

    i = 81;
    randperm(i, v);
    for(j = 0; j < 81; j++){
        while(v[j] < 0 || v[j] > 80){
            randperm(i, v);
        }
    }
    for(k = 0; k < n; k++){
        c[v[k]] = 0x1ff;
```

```
    }
    return;
}

int main()
{
    int i, j, k;
    int sol;
    int p;
    double x;
    char c[81];
    short s_c[81], c_c[81];

    set_rule();
    /*乱数初期化*/
    init_rnd((unsigned)time(NULL));

    make(c, s_c);

    for(k = 0; k < 81; k++){
        c_c[k] = x_c[k];
    }

    /*判定*/
    printf("穴の数 確率\n");
    for(i = 4; i < 65; i++){
        p = 0;
        for(j = 0; j < 200; j++){
            for(k = 0; k < 81; k++){
                s_c[k] = c_c[k];
                }
            sol = 0;
            boring(s_c, i);
            sol = search_sol(s_c, 2);
            if(sol == 1){
                p++;
            }
        }
        printf("%d %d\n", i, p);
    }
    return(0);
}
```