УДК 004.021
Коцюба А.Ю., Лавренчук С.В., Остапчук Р.В.
Луцький національний технічний університет

МЕТОДИКА ПОБУДОВИ АЛГОРИТМІВ ГЕНЕРАЦІЇ ВАРІАНТІВ ГОЛОВОЛОМКИ СУДОКУ

Коцюба А.Ю., Лавренчук С.В., Остапчук Р.В. Методика побудови алгоритмів генерації варіантів головоломки Судоку. Однією з найцікавіших задач з комбінаторики є задача про знаходження всіх можливих варіантів гри Судоку та її різновидів. Кількість таких варіантів є числом досить високого порядку, тому найбільш важливою є не так проблема пошуку всіх цих варіантів та їх кількості, як проблема розробки загальної методики побудови алгоритмів генерації різних варіантів головоломки Судоку. Зазвичай, ці алгоритми генерують не всю множину варіантів, а лише деяку її підмножину. При цьому важливими були б знання про всі додаткові обмеження, що призводять до появи цієї підмножини. Розроблена методика є такою, що її можна розвивати для будь-яких різновидів даної гри. Крім цього, перед тим як її застосовувати, необхідно, описати всі вхідні обмеження.
Ключові слова: латинський квадрат Ейлера, головоломка Судоку, задача з комбінаторики, VBA, web-гра Судоку, JavaScript.

Коцюба А.Ю., Лавренчук С.В., Остапчук Р.В. Методика построения алгоритмов генерации вариантов головоломки Судоку. Одной из наиболее интересных задач по комбинаторике является задача о нахождении всех возможных вариантов игры Судоку и ее разновидностей. Количество таких вариантов является числом достаточно высокого порядка, поэтому наиболее важной является не столько проблема поиска всех этих вариантов и их количества, как проблема разработки общей методики построения алгоритмов генерации различных вариантов головоломки Судоку. В большинстве случаев эти алгоритмы генерируют не всё множество вариантов, а только некоторое его подмножество. При этом важными были бы знания обо всех дополнительных ограничениях, которые приводят к появлению этого подмножества. Разработанная методика такова, что ее можно развивать для любых разновидностей данной игры. Кроме этого, перед тем как ее применять, необходимо, описать все входящие ограничения.
Ключевые слова: латинский квадрат Эйлера, головоломка Судоку, задача по комбинаторике, VBA, web-игра Судоку, JavaScript.

Kotsyuba A.Yu, Lavrenchuk S.V., Ostapchuk R.V. The technique of building algorithms generate variations Sudoku puzzles. One of the most interesting problems in combinatorics is the problem of finding all possible Sudoku game and its varieties. The number of such choices is the number of very high order, and most important is not the problem of finding all these options and their number, as the problem of developing a common technique of building algorithms generating various options sudoku puzzles. Typically, these algorithms do not generate a whole set of options, but only some subset of it. It would be important knowledge of all additional restrictions that lead to the emergence of this subset. The method is such that it can be developed for any species of game. In addition, before its use, it is necessary to describe all incoming limitations.
Keywords: latin square Euler, Sudoku puzzle, the problem of combinatorics, VBA, web-game Sudoku, JavaScript.

Вступ. Однією із найбільш популярних головололомок XXI ст. є Судоку. Її історія починається з праць відомого швейцарського математика, механіка та фізика Леонарда Ейлера (1707 – 1783 р.р.). У своїх працях він досліджував різновиди "магічних квадратів" з кількістю комірок 9, 16, 25 та 36. Також він займався досліженням "магічних квадратів", в яких у рядках та стовпцях не повторюються символи (спочатку це були літери латинського алфавіту). Такі квадрати називалися ще латинськими. У сучасному вигляді головоломку судоку вперше опублікували в 1979 році в журналі Word Games magazine. Автором головоломки був Гарвард Гаріс. Він використав принцип латинського квадрата Ейлера і застосував його в матриці розмірністю 9×9, додавши при цьому додаткові обмеження – цифри не повинні повторюватися у 9-ти внутрішніх квадратах розмірністю 3×3. В квітні 1984 року ця головоломка була опублікована видавництвом збірників різних головоломок, японською компанією Nicoly Inc., під заголовком "Число використовується лише один раз". Назву головоломці дав керівник видавництва – Kaji Maki. Su переклається як число, а docu – як єдине. 12 листопада 2004 року газета The Times вперше на своїх сторінках опублікувала головоломку Судоку. Ця публікація стала сенсацією, головоломка швидко поширилася по всій Британії, Австралії, Новій Зеландії та набула популярності у США.

З'явилося багато різновидів цієї головоломки (кількість найвідоміших сягає 30). Розглянемо коротко деякі з них:

1. Стрічки. У матриці порожніх комірок розмірністю 9×9 якимось (відмінним від білого) кольором, наприклад, сірим зафарбовуються групи по 7 комірок (рис. 1).

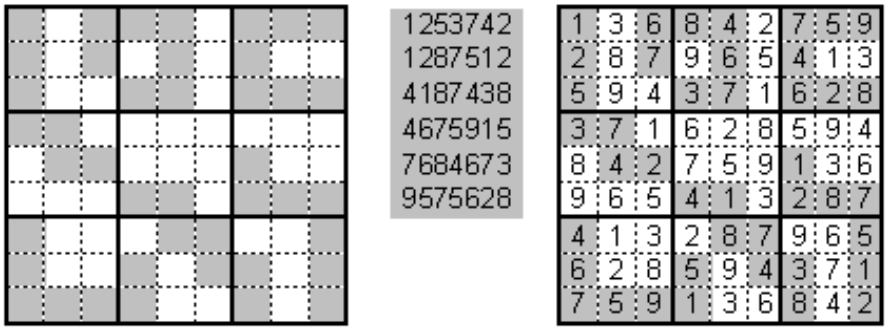

Рис. 1. Різновид головоломки Судоку – "Стрічки"

Дається набір цифр, яким необхідно заповнити зафарбовані комірки. Далі одержується судоку, яке необхідно розв'язати

2. Будівлі. Уявимо, що матриця з 81-ї комірки є скупченням такої ж кількості різноповерхових будинків, розташованих поруч один біля одного. При цьому спостерігач дивиться на ці будівлі зверху і нехай число, написане "на даху будинка", означає кількість його поверхів. Поблизу комплексу бувель вештаються інші спостерігачі. Обходячи даний комплекс з різних сторін вони можуть бачити лише ті будинки, які не закриті вищим будинком. Відповідно і з'являються числа, які означають кількість видимих з деякої сторони будівель (рис. 2).

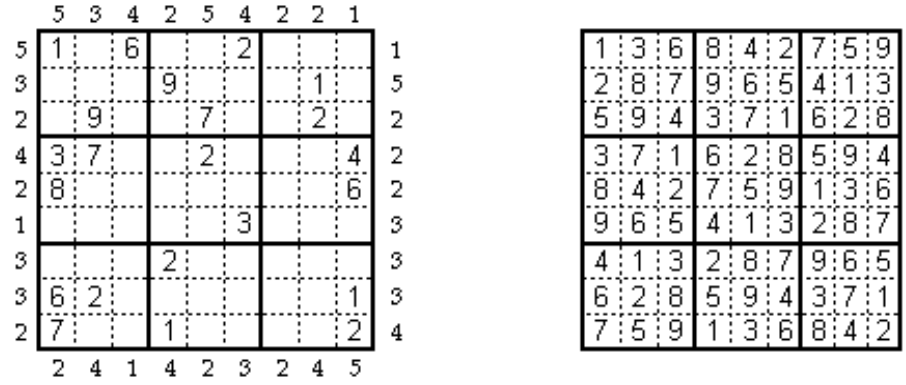

Рис. 2. Різновид головоломки Судоку – "Будівлі"

Всі інші правила заповнення є стандартними для Судоку.

3. Діагональна Судоку. Це різновид Судоку, який зустрічається найчастіше. Для його вирішення використовують стандартні правила для судоку. Відмінність від звичайного полягає лише в тому, що додається ще одне обмеження – цифри не повинні повторюватися по відмічених діагоналях (рис. 3).

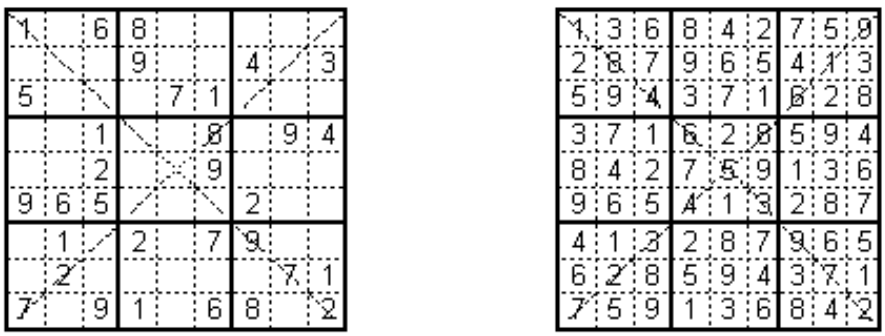

Рис. 3. Різновид головоломки Судоку з додатковими обмеженнями – "Діагональна Судоку"

Це були різновиди Судоку (не всі, а лише, на нашу думку, найвідоміші), в основі яких лежить правило, сформульоване Гарвардом Гарісом (надалі такі різновиди називатимемо класичними різновидами Судоку). Бувають ще й такі різновиди, у яких комірки розташовані взагалі не за

Гарісом (некласичні різновиди). Спільним у всіх цих головоломках залишається лише те, що закладено у їх назві – "Єдине число".

Постановка наукової проблеми та її значення. З появою таких головолок, виникла досить цікава, і на нашу думку, не досліджена до кінця проблема генерації різних варіантів головоломки Судоку. Нехай ми маємо справу з самим першим класичним різновидом цієї головоломки, запропонованим Гарвардом Гарісом (класичне судоку). На даний час не існує алгоритму, який би зміг згенерувати довільний варіант з множини всіх можливих варіантів цієї головоломки. Для побудови такого алгоритму необхідно було б спочатку розв'язати одну з нерозв'язаних, на нашу думку, задач з комбінаторики: "Скільки всього є варіантів класичної гри Судоку". Вирішення цієї задачі дозволить генерувати не лише варіанти цієї гри, а й варіанти класичних різновидів головоломки Судоку таких як: "Стрічки", "Будівлі" тощо. Крім того якщо розробити загальну методику, яка б дозволяла ще знаходити всі можливі варіанти для випадків класичних різновидів з додатковими обмеженнями – наприклад, діагональна Судоку тощо, то можна було б розпочати роботу над загальним алгоритмом, який дав би змогу розв'язувати відповідні комбінаторні задачі некласичних різновидів Судоку. І як результат – створити програмний комплекс, який би генерував різні варіанти різновидів гри Судоку. Це дало б можливість досліджувати алгоритми розв'язування різновидів Судоку, шукати мінімальну кількість підказок (ключів) не лише для класичної гри Судоку, а й для інших некласичних її різновидів.

Власне розробці такої методики і присвячена дана праця.

Аналіз останніх досліджень та публікацій. Більше дізнатися про латинський квадрат Ейлера можна з електронного ресурсу [1]. У джерелах [2, 3] наведено найвідоміші класичні та некласичні різновиди судоку. Варіант одного з найскладніших судоку наведено у праці [4]. Інформацію про мінімальну кількість підказок (ключів) для того, щоб судоку було задано однозначно можна знайти в електронних ресурсах [5, 6]. Для дослідження методик вирішення судоку можна використати джерела [3, 7–9]. Висновки про сучасні алгоритми генерації варіантів головоломки Судоку можна зробити з робіт [10–12]. Для вивчення технологій написання програм мовою VBA та скриптів мовою JavaScript можна використати праці [13] та [14] відповідно.

Мета роботи – розробка методики написання алгоритмів генерації різних варіантів головоломки Судоку. Слід зауважити, що мету можна розділити на початкову (власне яка і є основною вищесформульованою метою роботи) та кінцеву (єдина проблема, яка стоїть на заваді її практичної реалізації – це обчислювальна швидкодія сучасних персональних комп'ютерів). Кінцевою метою роботи є написання алгоритму генерації всіх можливих варіантів класичної гри Судоку.

Розробка методики. Спочатку теоретично розробимо методику, яка в разі наявності необхідної обчилювальної швидкодії сучасних ПК дозволить розв'язати задачу з комбінаторики: "Скільки всього є варіантів класичної гри Судоку".

Очевидно, кількість всіх можливих варіантів класичного судоку не можна обчислювати за формулою

$$N = \prod_{i=1}^{9} i!. \qquad (1)$$

Досить часто в літературі чи в електронних ресурсах [5, 12] помилково вважають, що шукана кількість всіх можливих варіантів класичного судоку сягає порядку 10^{21} (неважко, побачити, що саме такого порядку є величина, що обчислюється за формулою (1)). Очевидно, що додаткові обмеження, які на квадрат Ейлера розмірністю 9×9 наклав Гарвард Гаріс (а саме, цифри не повинні повторюватися у 9-ти внутрішніх квадратах розмірністю 3×3) призведуть до суттєвого зниження порядку шуканої кількості. При цьому формула (1) взагалі ніяк не допоможе. Цікаве рішення цієї залачі наведене у праці [12]

$$2297902829591040 \approx 2.3\cdot 10^{15}.$$

Наведений результат важко без доведення прийняти на віру, хоча він є досить реалістичним. Навіть, якщо припустити, що ця кількість і є розв'язком вищесформульованої задачі з комбінаторики, то необхідно навчитися генерувати всі можливі варіанти цієї головоломки.

Для того щоб, всі цифри у 9-ти внутрішніх квадратах (назвемо їх блоками (рис. 1–3)) не повторювалися, можна заповнювати Судоку за такою схемою: беремо спочатку першу цифру і розкладаємо її по блоках так, щоб не порушити базові обмеження латинського квадрата (цифри повинні бути різними у кожному рядку та стовпці), далі аналогічно по комірках, які залишилися розкладаємо другу цифру і так далі аж до останньої 9-ї цифри, для якої позиції в блоках вже будуть однозначно визначені. Для більшої наочності подальших викладок введемо позначення для цих цифр, які подані на рис. 4.

a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	a_{32}	a_{13}	a_{23}	a_{33}
a_{41}	a_{51}	a_{61}	a_{42}	a_{52}	a_{62}	a_{43}	a_{53}	a_{63}
a_{71}	a_{81}	a_{91}	a_{72}	a_{82}	a_{92}	a_{73}	a_{83}	a_{93}
a_{14}	a_{24}	a_{34}	a_{15}	a_{25}	a_{35}	a_{16}	a_{26}	a_{36}
a_{44}	a_{54}	a_{64}	a_{45}	a_{55}	a_{65}	a_{46}	a_{56}	a_{66}
a_{74}	a_{84}	a_{94}	a_{75}	a_{85}	a_{95}	a_{76}	a_{86}	a_{96}
a_{17}	a_{27}	a_{37}	a_{18}	a_{28}	a_{38}	a_{19}	a_{29}	a_{39}
a_{47}	a_{57}	a_{67}	a_{48}	a_{58}	a_{68}	a_{49}	a_{59}	a_{69}
a_{77}	a_{87}	a_{97}	a_{78}	a_{88}	a_{98}	a_{79}	a_{89}	a_{99}

Рис. 4. Позначення для заповненого варіанту судоку

Згідно з наведеними на рис. 4 позначеннями для того, щоб в кожному блоці зустрічалася лише одна цифра, необхідно кожній цифрі поставити у відповідність деяке правило

$$\left(a_{k_1 1}, a_{k_2 2}, ..., a_{k_9 9}\right), \tag{2}$$

у якому $k_1, k_2, ..., k_9 = \overline{1,9}$. Сформулюємо ще такі умови на числа $k_1, k_2, ..., k_9$, щоб виконувалися базові обмеження (надалі так називатимемо обмеження, які були характерні ще для латинського квадрата Ейлера). Якщо множину цифр від *1* до *9* розбити на *3* підмножини: $R_1 = \{1,2,3\}$, $R_2 = \{4,5,6\}$ та $R_3 = \{7,8,9\}$; то першу умову (цифри повинні бути різними у кожному рядку) можна сформулювати наступним чином: для будь-яких кортежів $\left(k_i, k_j\right)$ таких, що $\left(i, j \in R_1\right) \vee$ $\vee \left(i, j \in R_2\right) \vee \left(i, j \in R_3\right)$ повинна виконуватися умова $\left(k_i, k_j \notin R_1\right) \wedge \left(k_i, k_j \notin R_2\right) \wedge \left(k_i, k_j \notin R_3\right)$. Для формулювання відповідної другої умови множину цифр від *1* до *9* необхідно розбити на *3* інші підмножини: $C_1 = \{1,4,7\}$, $C_2 = \{2,5,8\}$ та $C_3 = \{3,6,9\}$. Далі для будь-яких кортежів $\left(k_i, k_j\right)$ друга умова формулюється аналогічно. В середовищі MS Excel за допомогою VBA неважко написати макрос, який би видав всі можливі правила (2), які у спрощеній формі легше буде записати у вигляді

$$\left(k_1, k_2, ..., k_9\right). \tag{2'}$$

Покажемо два фрагменти коду відповідного макросу (рис. 5).

```
Sub Nabors_spivpad()
'
' Nabors_spivpad Макрос
' Макрос записан 16.10.2015 (SPA)
'
' Сочетание клавиш: Ctrl+d
'
  Dim lich As Integer
  Dim i1 As Integer
  Dim i2 As Integer
  Dim i3 As Integer
  Dim i4 As Integer
  Dim i5 As Integer
  Dim i6 As Integer
  Dim i7 As Integer
  Dim i8 As Integer
  Dim i9 As Integer
  lich = -32768
  For i1 = 1 To 9
    For i2 = 1 To 9
      For i3 = 1 To 9
        For i4 = 1 To 9
          For i5 = 1 To 9
            For i6 = 1 To 9
              For i7 = 1 To 9
                For i8 = 1 To 9
                  For i9 = 1 To 9
                    If ((i2 = 2) And (i3 = 3)) Then
                    ElseIf ((i2 = 3) And (i3 = 2)) Then
                    ElseIf ((i2 = 1) And (i3 = 1)) Then
                    ElseIf ((i2 = 1) And (i3 = 3)) Then
                    ElseIf ((i2 = 3) And (i3 = 1)) Then
                    ElseIf ((i2 = 2) And (i3 = 2)) Then
                    ElseIf ((i2 = 1) And (i3 = 2)) Then
                    ElseIf ((i3 = 6) And (i6 = 9)) Then
                    ElseIf ((i3 = 9) And (i6 = 6)) Then
                    ElseIf ((i3 = 3) And (i6 = 3)) Then
                    ElseIf ((i3 = 3) And (i6 = 9)) Then
                    ElseIf ((i3 = 9) And (i6 = 3)) Then
                    ElseIf ((i3 = 6) And (i6 = 6)) Then
                    ElseIf ((i3 = 3) And (i6 = 6)) Then
                    ElseIf ((i3 = 6) And (i6 = 3)) Then
                    ElseIf ((i3 = 9) And (i6 = 9)) Then
                    Else
                      lich = lich + 1
                      Cells(lich + 32768, 1) = i1
                      Cells(lich + 32768, 2) = i2
                      Cells(lich + 32768, 3) = i3
                      Cells(lich + 32768, 4) = i4
                      Cells(lich + 32768, 5) = i5
                      Cells(lich + 32768, 6) = i6
                      Cells(lich + 32768, 7) = i7
                      Cells(lich + 32768, 8) = i8
                      Cells(lich + 32768, 9) = i9
                    End If
                  Next i9
                Next i8
              Next i7
            Next i6
          Next i5
        Next i4
      Next i3
    Next i2
  Next i1
End Sub
```

Рис. 5. Код макросу (початок і кінець), який видає всі можливі правила у вигляді (2')

Тепер залишається показати також у скороченому вигляді відповідний результат (рис. 6).

	A	B	C	D	E	F	G	H	I	J
1	1	1	4	7	2	5	8	3	6	9
2	2	1	4	7	2	5	8	3	9	6
3	3	1	4	7	2	5	8	6	3	9
4	4	1	4	7	2	5	8	6	9	3
5	5	1	4	7	2	5	8	9	3	6
6	6	1	4	7	2	5	8	9	6	3
7	7	1	4	7	2	5	9	3	6	8
8	8	1	4	7	2	5	9	3	9	5
9	9	1	4	7	2	5	9	6	3	8
10	10	1	4	7	2	5	9	6	9	2
11	11	1	4	7	2	5	9	9	3	5
12	12	1	4	7	2	5	9	9	6	2
13	13	1	4	7	2	6	8	3	5	9
14	14	1	4	7	2	6	8	3	8	6
15	15	1	4	7	2	6	8	6	2	9
16	16	1	4	7	2	6	8	6	8	3
17	17	1	4	7	2	6	8	9	2	6
18	18	1	4	7	2	6	8	9	5	3
19	19	1	4	7	2	6	9	3	5	8
20	20	1	4	7	2	6	9	3	8	5
21	21	1	4	7	2	6	9	6	2	8

	A	B	C	D	E	F	G	H	I	J
46636	46636	9	6	3	8	4	1	4	8	2
46637	46637	9	6	3	8	4	1	7	2	5
46638	46638	9	6	3	8	4	1	7	5	2
46639	46639	9	6	3	8	4	2	1	5	7
46640	46640	9	6	3	8	4	2	1	8	4
46641	46641	9	6	3	8	4	2	4	2	7
46642	46642	9	6	3	8	4	2	4	8	1
46643	46643	9	6	3	8	4	2	7	2	4
46644	46644	9	6	3	8	4	2	7	5	1
46645	46645	9	6	3	8	5	1	1	4	8
46646	46646	9	6	3	8	5	1	1	7	5
46647	46647	9	6	3	8	5	1	4	1	8
46648	46648	9	6	3	8	5	1	4	7	2
46649	46649	9	6	3	8	5	1	7	1	5
46650	46650	9	6	3	8	5	1	7	4	2
46651	46651	9	6	3	8	5	2	1	4	7
46652	46652	9	6	3	8	5	2	1	7	4
46653	46653	9	6	3	8	5	2	4	1	7
46654	46654	9	6	3	8	5	2	4	7	1
46655	46655	9	6	3	8	5	2	7	1	4
46656	46656	9	6	3	8	5	2	7	4	1

Рис. 6. Результат роботи вищенаведеного макросу

З цього рисунку очевидно, що всього є *46656* правил виду (2'), за допомогою яких можна скласти всі варіанти класичного судоку. При складанні варіантів судоку слід зазначити, що не всі правила можна поєднувати. Тут мається на увазі, що якщо першій цифрі поставити у відповідність одне з *46656* правил, то другій цифрі можна ставити у відповідність будь-яке правило з множини цих правил, яке "немає нічого спільного з попереднім правилом". А тепер пояснимо, що означає вислів взятий у лапки.

Означення 1. Два правила виду (2') будемо називати *можливими сусідами* або такими, що *не мають нічого спільного*, якщо на однакових позиціях не має однакових цифр. Такі правила надалі коротко називатимемо *сусідними*.

Очевидно, що якщо скласти матрицю розмірністю 9×9 таку, щоб номер рядка був би, наприклад, цифрою, а номер стовпця – позицією, то в кожну комірку такої матриці можна було б записати множини всіх можливих правил, у яких на відповідній позиції зустрічається відповідна

цифра. І якщо потужності цих множин будуть однакові, то можна припустити, що програма дала правильний результат. Неважко методом від супртивного показати, що цей критерій є необхідною умовою повноти шуканої множини правил. І для вищеописаних *46656* правил виду (2'), було показано, що такі потужності в кожній комірці матриці розмірності 9×9 є однаковими і становлять $46656/9 = 5184$.

Для того, щоб знайти кількість всіх можливих варіантів класичного судоку, нам необхідно знайти множину всіх множин (наборів) з 9 правил, які попарно є можливими сусідами. Якщо потужність такої множини позначимо, через N_{nab}, то, очевидно, розв'язок розглянутої задачі з комбінаторики буде знаходитися за формулою

$$N_{sudocu} = N_{nab} \cdot 9!, \tag{3}$$

тобто для генерації всіх можливих варіантів ігор, в програму необхідно внести:

- *46656* правил (2'), частина з яких нами наведена на рис. 6;
- масив, елементами, якого є *різні* відсортовані в порядку, наприклад, зростання набори (масиви) з 9-ти правил, кожні два з яких є можливими сусідами (цей масив повинен містити N_{nab} наборів).

Якщо припустити, що наведена у праці [12] кількість $N_{sudocu} = 2297902829591040$, то N_{nab} не буде натуральним числом, тобто можна вважати, що вищесформульована задача з комбінаторики не є розв'язаною.

Далі залишається лише за допомогою ГПВЧ згенерувати 2 числа:

- перше число (з множини $\{0,1,...,N_{nab}-1\}$) буде задавати набір;
- друге число (з множини $\{0,1,...,9!-1\}$) буде задавати порядок, тобто номер правила для цифри 1, номер правила для цифри 2 і т.д.

Отже, розв'язання даної комбінаторної задачі зводиться лише до пошуку всіх можливих різних та відсортованих наборів із сусідніх правил та підрахунку кількості цих наборів.

Для вирішення цієї проблеми пропонуємо наступний алгоритм:

1 етап. Відсортуємо та прономеруємо правила (як на рис. 6). Розіб'ємо їх (надалі набір будемо ототожнювати з номером, який йому відповідає) на 9 підножин, таких, що в них жодні два правила не можуть бути сусідніми. Зробимо це, наприклад, по порядку

$$\{1,2,...,5184\},\ \{5185,5186,...,10368\},\ \{10369,10370,...,15552\},$$
$$\{15553,15554,...,20736\},\ \{20737,20738,...,25920\},\ \{25921,25922,...,31104\},$$
$$\{31105,31106,...,36288\},\ \{36289,36290,...,41472\},\ \{41473,41474,...,46656\}. \tag{4}$$

2 етап. Для кожного правила внесемо масив всіх можливих, наприклад, несусідніх по відношенню до нього правил. Можна брати і масив сусідніх правил, просто при цьому дещо зміняться наступні етапи алгоритму. Доречі, для даної задачі, виписати всіх сусідів, чи не сусідів, досить трудоємкий процес навіть для потужних ПК. Якщо з множини всіх несусідніх правил виключити ті правила, які входять в одну і ту ж підмножину (4), то одержимо, що для кожного фіксованого правила буде 23500 несусідніх правил. А тепер уявіть матрицю розмірності 46656×23500 чисел від *1* до *46656*. Хоча визначення елементів даного масиву може зайняти якийсь час, це не найбільша проблема, яка стоїть на заваді пошуку відсортованих різних наборів.

3 етап. Далі залишається перебирати правила з першої підмножини (4). При кожному такому виборі перебирати правила з другої підмножини (4) і т.д. Для кожних з 9-ти вибраних правил попарно перевіряти, чи може серед них знайтися хоча б 2 несусідніх правила (для перевірки використовуємо вищесформований масив несусідніх правил). Якщо несусідніх правил немає, то такий набір записуємо, в протилежному випадку рухаємося далі.

Розбиття на підмножини (4) має свої вагомі переваги. Одержані в результаті перебору набори автоматично стають відсортованими і як результат – не може з'явитися ще один набір рівний (з точністю до перемішування) якомусь набору, який був уже знайдений.

Виникає логічне запитання: яка проблема не дозволить програмно реалізувати вищеописаний алгоритм? А проблема полягає у тому, що для повного перебору необхідно проробити $5184^9 \cdot 23500$ операцій. Залишається лише справа за малим: вибрати середовище та обчислювальні технології, які б дозволили за реальний проміжок часу проробити таку кількість операцій.

Апробація розробленої методики. У зв'язку з вищеописаними проблемами, нами було вирішено накласти додаткові обмеження: вимоги для виконання базових обмежень перемішати і додати обмеження про те, що $k_1, k_2, ..., k_9$ є різними, тобто добавити ще три умови:

- для будь-яких кортежів (k_i, k_j) таких, що $(i, j \in R_1) \vee (i, j \in R_2) \vee (i, j \in R_3)$ повинна виконуватися умова $(k_i, k_j \notin C_1) \wedge (k_i, k_j \notin C_2) \wedge (k_i, k_j \notin C_3)$;
- для будь-яких кортежів (k_i, k_j) таких, що $(i, j \in C_1) \vee (i, j \in C_2) \vee (i, j \in C_3)$ повинна виконуватися умова $(k_i, k_j \notin R_1) \wedge (k_i, k_j \notin R_2) \wedge (k_i, k_j \notin R_3)$;
- $k_1 \neq k_2 \neq ... \neq k_9$.

В результаті таких обмежень одержимо 144 правила. На першому етапі алгоритму множину правил розіб'ємо аналогічно до сукупності (4) на такі підмножини:

$$\{1,2,...,16\},\ \{17,18,...,32\},\ \{33,34,...,48\},$$
$$\{49,50,...,64\},\ \{65,66,...,80\},\ \{81,82,...,96\},$$
$$\{97,98,...,112\},\ \{113,114,...,128\},\ \{129,130,...,144\}. \quad (4')$$

Застосувавши вищеописаний алгоритм до множини цих правил та до сукупності підмножин (4'), одержимо 664 набори. За формулою (3) неважко порахувати, що таким чином, можна побудувати алгоритм, який генеруватиме $664 \cdot 9! = 240952320$ варіантів гри класичного судоку. Що і було зроблено засобами web-програмування. Покажемо фрагмент коду мовою JavaScript, у якому введено 144 правила та 664 набори з 9-ти сусідніх правил (рис. 7).

```
var pravylos=new Array(144);
pravylos[0]=[1,5,9,2,6,7,3,4,8]; pravylos[1]=[1,5,9,3,4,8,2,6,7]; pravylos[2]=[1,5,9,6,7,2,8,3,4]; pravylos[3]=[1,5,9,8,3,4,6,7,2]; pravylos[4]=[1,6,8,2,4,9,3,5,7]; pravylos[5]
pravylos[16]=[2,4,9,1,6,8,3,5,7]; pravylos[17]=[2,4,9,3,5,7,1,6,8]; pravylos[18]=[2,4,9,6,8,1,7,3,5]; pravylos[19]=[2,4,9,7,3,5,6,8,1]; pravylos[20]=[2,6,7,1,5,9,3,4,8]; pravy
pravylos[32]=[3,4,8,1,5,9,2,6,7]; pravylos[33]=[3,4,8,2,6,7,1,5,9]; pravylos[34]=[3,4,8,5,9,1,7,2,6]; pravylos[35]=[3,4,8,7,2,6,5,9,1]; pravylos[36]=[3,5,7,1,6,8,2,4,9]; pravy
pravylos[48]=[4,2,9,3,7,5,8,6,1]; pravylos[49]=[4,2,9,5,3,7,6,1,8]; pravylos[50]=[4,2,9,6,1,8,5,3,7]; pravylos[51]=[4,2,9,8,6,1,3,7,5]; pravylos[52]=[4,3,8,2,7,6,9,5,1]; pravy
pravylos[64]=[5,1,9,3,8,4,7,6,2]; pravylos[65]=[5,1,9,4,3,8,6,2,7]; pravylos[66]=[5,1,9,6,2,7,4,3,8]; pravylos[67]=[5,1,9,7,6,2,3,8,4]; pravylos[68]=[5,3,7,1,8,6,9,4,2]; pravy
pravylos[80]=[6,1,8,2,9,4,7,5,3]; pravylos[81]=[6,1,8,4,2,9,5,3,7]; pravylos[82]=[6,1,8,5,3,7,4,2,9]; pravylos[83]=[6,1,8,7,5,3,2,9,4]; pravylos[84]=[6,2,7,1,9,5,8,4,3]; pravy
pravylos[96]=[7,2,6,3,4,8,5,9,1]; pravylos[97]=[7,2,6,5,9,1,3,4,8]; pravylos[98]=[7,2,6,8,3,4,9,1,5]; pravylos[99]=[7,2,6,9,1,5,8,3,4]; pravylos[100]=[7,3,5,2,4,9,6,8,1]; pravy
pravylos[112]=[8,1,6,3,5,7,4,9,2]; pravylos[113]=[8,1,6,4,9,2,3,5,7]; pravylos[114]=[8,1,6,7,3,5,9,2,4]; pravylos[115]=[8,1,6,9,2,4,7,3,5]; pravylos[116]=[8,3,4,1,5,9,6,7,2];
pravylos[128]=[9,1,5,2,6,7,4,8,3]; pravylos[129]=[9,1,5,4,8,3,2,6,7]; pravylos[130]=[9,1,5,7,2,6,8,3,4]; pravylos[131]=[9,1,5,8,3,4,7,2,6]; pravylos[132]=[9,2,4,1,6,8,5,7,3];
var nabors=new Array(664);
nabors[0]=[1,22,33,58,79,90,99,120,131]; nabors[1]=[1,22,33,58,79,90,100,119,132]; nabors[2]=[1,22,33,58,79,90,103,116,135]; nabors[3]=[1,22,33,58,79,90,104,115,136]; nabors[4]
nabors[13]=[1,22,33,63,74,95,100,119,132]; nabors[14]=[1,22,33,63,74,95,103,116,135]; nabors[15]=[1,22,33,63,74,95,104,115,136]; nabors[16]=[1,22,35,60,79,90,99,117,131]; nabo
nabors[26]=[1,24,33,58,77,90,99,118,131]; nabors[27]=[1,24,36,58,77,89,99,118,130]; nabors[28]=[1,24,41,58,77,82,99,118,139]; nabors[29]=[1,24,43,58,77,84,99,118,137]; nabors[
nabors[39]=[1,30,36,58,71,89,99,128,130]; nabors[40]=[1,30,41,58,71,82,99,128,139]; nabors[41]=[1,30,43,58,71,84,99,128,137]; nabors[42]=[1,31,33,58,72,90,99,125,131]; nabors[
nabors[46]=[2,21,34,58,79,90,99,120,131]; nabors[47]=[2,21,34,58,79,90,100,119,132]; nabors[48]=[2,21,34,58,79,90,103,116,135]; nabors[49]=[2,21,34,58,79,90,104,115,136]; nabo
nabors[59]=[2,21,34,63,74,95,100,119,132]; nabors[60]=[2,21,34,63,74,95,103,116,135]; nabors[61]=[2,21,34,63,74,95,104,115,136]; nabors[62]=[2,21,35,59,78,92,100,119,129]; nabo
nabors[72]=[2,24,34,59,78,89,98,119,132]; nabors[73]=[2,24,36,57,78,89,98,118,132]; nabors[74]=[2,24,46,55,78,89,98,123,132]; nabors[75]=[2,24,48,53,78,89,98,122,132]; nabors[
nabors[85]=[2,29,35,59,70,92,100,127,129]; nabors[86]=[2,29,42,59,70,83,100,127,140]; nabors[87]=[2,29,44,59,70,81,100,127,138]; nabors[88]=[2,32,34,59,69,91,100,126,132]; nabo
nabors[92]=[3,21,34,60,78,91,97,119,132]; nabors[93]=[3,21,35,60,78,92,97,119,129]; nabors[94]=[3,21,42,60,78,83,97,119,140]; nabors[95]=[3,21,44,60,78,81,97,119,138]; nabors[
nabors[105]=[3,23,35,64,76,96,101,113,133]; nabors[106]=[3,23,36,60,77,92,98,117,129]; nabors[107]=[3,23,42,60,80,83,97,117,140]; nabors[108]=[3,23,44,60,80,81,97,117,138]; nab
nabors[118]=[3,27,35,60,80,88,109,117,129]; nabors[119]=[3,27,45,54,80,88,109,124,129]; nabors[120]=[3,27,47,56,80,88,109,121,129]; nabors[121]=[3,29,34,60,70,91,97,127,132];
nabors[129]=[4,21,34,59,77,91,100,119,130]; nabors[130]=[4,21,36,57,77,91,100,118,130]; nabors[131]=[4,21,46,55,77,91,100,123,130]; nabors[132]=[4,21,48,53,77,91,100,122,130];
nabors[142]=[4,24,36,57,77,89,98,118,130]; nabors[143]=[4,24,36,57,77,89,102,114,134]; nabors[144]=[4,24,36,61,73,93,98,118,130]; nabors[145]=[4,24,36,61,73,93,102,114,134]; na
nabors[155]=[4,28,36,57,77,85,110,118,130]; nabors[156]=[4,28,46,55,77,85,110,123,130]; nabors[157]=[4,28,48,53,77,85,110,122,130]; nabors[158]=[4,30,33,57,71,90,98,128,131];
nabors[166]=[5,18,37,58,79,90,99,120,131]; nabors[167]=[5,18,37,58,79,90,100,119,132]; nabors[168]=[5,18,37,58,79,90,103,116,135]; nabors[169]=[5,18,37,58,79,90,104,115,136];
nabors[179]=[5,18,37,63,74,95,100,119,132]; nabors[180]=[5,18,37,63,74,95,103,116,135]; nabors[181]=[5,18,37,63,74,95,104,115,136]; nabors[182]=[5,18,39,62,75,96,103,116,133];
nabors[192]=[5,20,37,61,75,94,103,116,134]; nabors[193]=[5,20,40,61,73,94,102,116,134]; nabors[194]=[5,20,41,61,71,94,107,116,134]; nabors[195]=[5,20,44,61,69,94,106,116,134];
nabors[205]=[5,30,40,50,73,94,102,116,139]; nabors[206]=[5,30,41,50,71,94,107,116,139]; nabors[207]=[5,30,44,50,69,94,106,116,139]; nabors[208]=[5,32,37,49,75,94,103,116,138];
nabors[212]=[6,17,38,58,79,90,99,120,131]; nabors[213]=[6,17,38,58,79,90,100,119,132]; nabors[214]=[6,17,38,58,79,90,103,116,135]; nabors[215]=[6,17,38,58,79,90,104,115,136];
nabors[225]=[6,17,38,63,74,95,100,119,132]; nabors[226]=[6,17,38,63,74,95,103,116,135]; nabors[227]=[6,17,38,63,74,95,104,115,136]; nabors[228]=[6,17,39,63,76,95,101,115,136];
nabors[238]=[6,20,38,63,73,95,104,114,136]; nabors[239]=[6,20,40,63,73,93,104,114,134]; nabors[240]=[6,20,46,63,73,87,104,114,144]; nabors[241]=[6,20,47,63,73,88,104,114,141];
nabors[251]=[6,29,39,51,76,95,101,115,140]; nabors[252]=[6,29,42,51,70,95,108,115,140]; nabors[253]=[6,29,43,51,72,95,105,115,140]; nabors[254]=[6,31,38,52,74,95,104,115,137];
nabors[258]=[7,17,38,63,74,96,104,113,136]; nabors[259]=[7,17,39,63,76,96,101,113,136]; nabors[260]=[7,17,42,63,70,96,108,113,136]; nabors[261]=[7,17,43,63,72,96,105,113,136];
nabors[271]=[7,19,39,64,76,96,101,113,133]; nabors[272]=[7,19,40,61,76,96,101,114,133]; nabors[273]=[7,19,42,64,70,96,108,113,133]; nabors[274]=[7,19,43,64,72,96,105,113,133];
```

Рис. 7. Частина правил та наборів, отриманих при реалізації вищеописаної методики

Далі покажемо, як відбувається генерація випадкового набору:

- нехай з множини $\{0,1,...,663\}$ наш ГПВЧ згенерував число *213*;

- нехай з множини $\{0,1,...,9!-1\}$ наш ГПВЧ згенерував число *100330*.

Тоді з першого згенерованого числа *213* одержимо (із врахуванням зміщення на 1 в індексах) наступні набор та відповідні правила (рис. 8).

```
nabors[213]=[6,17,38,58,79,90,100,119,132];
pravylos[5]=  [1,6,8,3,5,7,2,4,9];
pravylos[16]= [2,4,9,1,6,8,3,5,7];
pravylos[37]= [3,5,7,2,4,9,1,6,8];
pravylos[57]= [4,8,3,5,9,1,6,7,2];
pravylos[78]= [5,9,1,6,7,2,4,8,3];
pravylos[89]= [6,7,2,4,8,3,5,9,1];
pravylos[99]= [7,2,6,9,1,5,8,3,4];
pravylos[118]=[8,3,4,7,2,6,9,1,5];
pravylos[131]=[9,1,5,8,3,4,7,2,6];
```

Рис. 8. Випадково згенерований набір та відповідні йому правила

Далі покажемо, що завдяки другому згенерованому числу *100330* кортеж $(1,2,3,4,5,6,7,8,9)$ перетвориться в кортеж $(5,8,1,4,2,6,9,7,3)$, тобто набір $(6,17,38,58,79,90,100,119,132)$ перетвориться в набір $(79,119,6,58,17,90,132,100,38)$. В результаті у відповідній web-реалізації розробленої методики одержимо випадок класичного судоку (рис. 9).

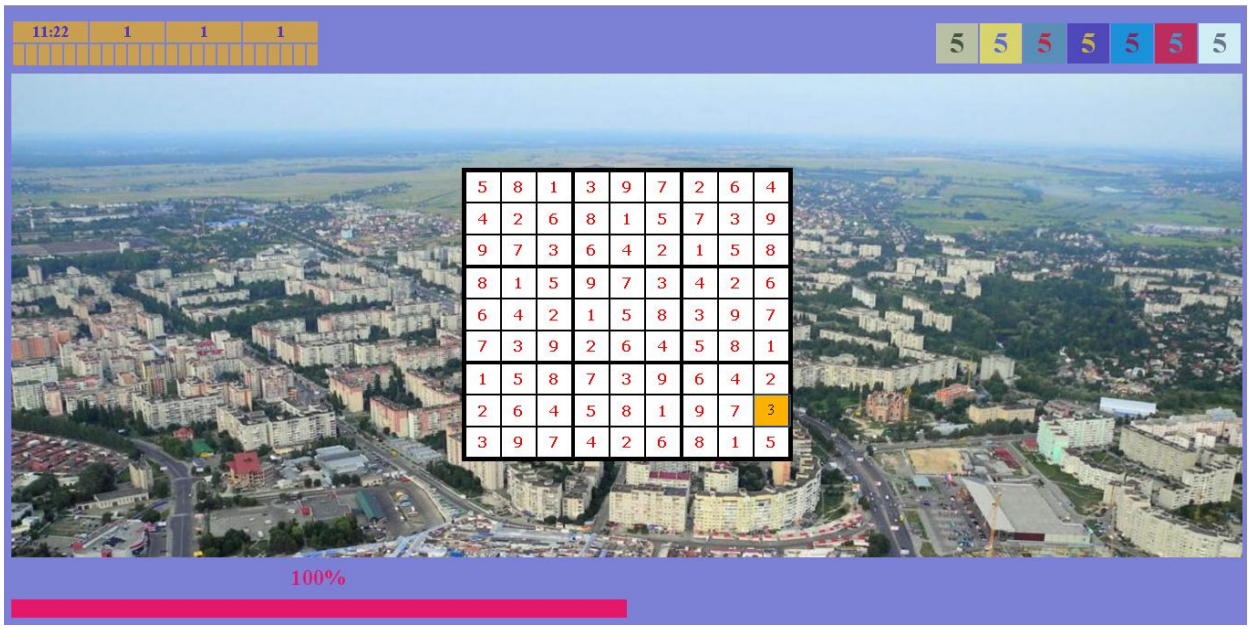

Рис. 9. Приклад web-реалізації розробленої методики генерації варіантів класичної гри Судоку

Далі покажемо, як для вищеописаних параметрів $(213;100330)$ відбувається дві випадкові генерації з 25-ма ключами (рис. 10)

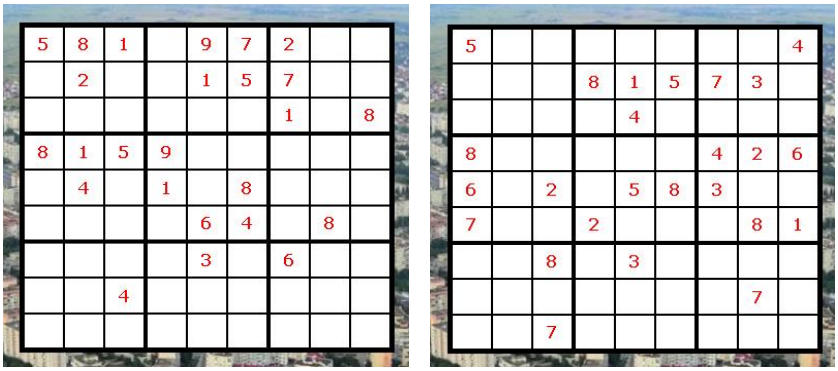

Рис. 10. Два приклади головоломки Судоку, зображеного на рис. 9, з *25*-ма ключами

Висновки. Отже, початкову мету роботи реалізовано. Щодо реалізації кінцевої мети, то тут необхідно ще попрацювати в наступних напрямках:

- по-перше, можна накласти не такі сильні обмеження (наприклад, можна спробувати дослідити класичний різновид даної головоломки – діагональне судоку);
- по-друге, у будь-якому випадку необхідно спробувати одні із найшвидших для таких задач мови програмування C++ або Fortran (при цьому важливо навчитися розподіляти операції між ядрами комп'ютерів локальної мережі);
- по-третє, є некласичні різновиди Судоку, вирішення комбінаторної задачі для яких може потребувати значно меншої кількості операцій.

Хоча в даній роботі нами не було знайдено за формулою (3) число N_{sudocu} всіх можливих варіантів класичної гри Судоку, але все ж таки ефективну методику, яку можна розвинути для некласичних різновидів цієї головоломки було розроблено, тобто з основною (не кінцевою) метою роботи справилися.

1. Википедия: Свободная энциклопедия [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/Латинский_квадрат.
2. Википедия: Свободная энциклопедия [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/Судоку.
3. Судоку-клуб [Електронний ресурс]. – Режим доступу : http://www.sudoku-club.ru/variations.html.
4. Попробуйте решить самую трудную головоломку в мире [Електронний ресурс]. – Режим доступу : http://tengrinews.kz/progress/poprobuyte-reshit-samuyu-trudnuyu-golovolomku-v-mire-216712/index.php.
5. Lenta.ru: Наука и техника. – Математики решили задачу о подсказках в судоку [Електронний ресурс]. – Режим доступу : http://lenta.ru/news/2012/01/09/sudoku/index.php.
6. ВКонтакте: Социальная сеть [Електронний ресурс]. – Режим доступу : https://vk.com/topic-1196-3359_27503324.
7. Судоку и какуро. Решебник [Електронний ресурс]. – Режим доступу : http://su-doku.ru/puzzle/-standart/index.php.
8. Хабрахабр. Методы решения судоку [Електронний ресурс]. – Режим доступу : http://habrahabr.ru/-post/173795/index.php.
9. Интернет-журнал "KM.RU". – Математики придумали формулу для решения судоку [Електронний ресурс]. – Режим доступу : http://www.km.ru/science-tech/2012/10/19/issledovaniya-rossiiskikh-i-zarubezhnykh-uchenykh/695292-matematiki-pridumal.
10. Хабрахабр. Алгоритм генерации судоку [Електронний ресурс]. – Режим доступу : http://habrahabr.ru/post/192102/index.php.
11. Интернет-журнал "IT race". – Генератор игрового поля в Судоку [Електронний ресурс]. – Режим доступу : http://it-race.blogspot.com/2012/02/blog-post_29.html.
12. Василенко С.Л. Числовая гармлния Судоку / С.Л. Василенко // Научно-технический форум : SciTecLibrary.ru [Електронний ресурс]. – 2012. – Режим доступу : http://www.sciteclibrary.ru.
13. Каленюк П.І. Практикум з програмування на VBA / П.І. Каленюк, А.Ф. Обшта, Н.М. Гоблик та ін. – Львів : Видавництво Національного університетуту "Львівська політехніка", 2005. – 208 с.
14. Флэнаган Д. JavaScript. Подробное руководство / Д. Флэнаган. – СПб. : "Символ-Плюс", 2008. – 992 с.