*Пример использования алгоритма решения n-Queens Completion Problem.*

*Григорян Э.М.*

1. *Генерация композиции*

В программе *Generation\_k\_Queens\_Composition* присвоим *n = 100*, чтобы рассмотреть шахматную доску размером *100 x 100* . После запуска программы, будет сформирована некоторая произвольная композиция. Пусть, в результате была получена композиция из *47* ферзей: *Q(1:100) =*

0 44 16 33 77 0 0 52 30 0

0 0 91 0 66 31 0 0 0 0

0 56 53 59 28 22 39 0 0 0

90 0 0 0 37 0 97 99 24 87

38 49 0 94 0 74 0 0 0 0

65 0 47 0 92 0 0 0 0 7

69 80 0 0 0 0 0 35 88 76

0 29 0 1 18 50 19 0 0 0

68 0 0 10 43 0 0 0 100 61

8 0 0 60 41 0 0 0 0 0

Сто чисел, которые перечислены выше, последовательно характеризуют *100* строк матрицы решения. Значение ноль указывает, что в соответствующей строке ферзь не установлен (строка свободна), любое иное число указывает позицию ферзя в рассматриваемой строке. Например, в полученной композиции первый элемент массива *Q(1:100)* равен *0*, это означает, что первая строка шахматной доски свободна; второй элемент массива равен *44*, это означает, что во второй строке шахматной доски ферзь расположен в позиции *44*; и т.д.

Сохраним полученные данные в файле *kQueens\_Test\_Composition.mat*. Такое имя указано в конце программы *Generation\_k\_Queens\_Composition*  в качестве примера. (Очевидно, что можно указать любое другое имя)

1. *Комплектация композиции до полного решения*

Для комплектации полученной композиции, запустим на исполнение программу *Solution\_n\_Queens\_Completion\_Problem*, где следует указать имя входного файла с композицией. (В данном примере мы используем имя входного файла данных: *kQueens\_Test\_Composition.mat*) В результате, рассматриваемая композиция будет комплектована, и полученные результаты будут сохранены в файле: *nQueens\_Test\_Completion\_Solution.mat* .(Данное имя файла выбрана в качестве примера). Полученное решение имеет вид:

>> Solution \_nQueens\_Completion\_Problem

Input file name: kQueens\_Test\_Composition.mat

The size of a chessboard= 100

Composition Size = 47

Number of free Positions = 53

The first 50 positions of queens:

0 44 16 33 77 0 0 52 30 0

0 0 91 0 66 31 0 0 0 0

0 56 53 59 28 22 39 0 0 0

90 0 0 0 37 0 97 99 24 87

38 49 0 94 0 74 0 0 0 0

Elapsed time is 0.051897 seconds.

Number of complete re-counting cycles = 0

Total number of usage the Back Tracking procedure = 0

Solution is Ok!

The first 50 positions of solution:

58 44 16 33 77 26 63 52 30 34

20 57 91 54 66 31 93 71 55 21

84 56 53 59 28 22 39 95 51 25

90 81 3 75 37 79 97 99 24 87

38 49 11 94 6 74 40 4 12 15

65 36 47 2 92 73 67 27 85 7

69 80 14 89 96 5 83 35 88 76

48 29 13 1 18 50 19 82 32 17

68 70 72 10 43 46 86 78 100 61

8 64 98 60 41 45 9 42 23 62

Solution saved in file: nQueens\_Test\_Completion\_Solution.mat

1. *Проверка правильности решения, полученного после комплектации*

Чтобы проверить правильность полученного решения воспользуемся программой *Validation\_n\_Queens\_Problem\_Solution*. Здесь, в качестве имени входного файла указано: *nQueens\_Test\_Completion\_Solution.mat* . (Если будет рассматриваться другой файл, то необходимо указать соответствующее имя.) После запуска, программа выведет следующее сообщение:

>> Validation\_n\_Queens\_Problem\_Solution

Input file name: nQueens\_Test\_Completion\_Solution.mat

The size of a chessboard = 100

Elapsed time is 0.000021 seconds.

Solutions is ok!

1. *Проверка правильности композиции*

Как мы уже говорили, программа *Validation\_n\_Queens\_Problem\_Solution* позволяет не только проверять правильность решения *n-Queens Problem*, но и правильность произвольной композиции.

Для проверки правильности полученной композиции запустим на исполнение программу *Validation\_n\_Queens\_Problem\_Solution*, где следует указать имя входного файла с композицией. ( Здесь, в качестве входного файла данных указано имя *kQueens\_Test\_Composition.mat*, которое мы использовали для сохранения композиции ).

В результате проверки рассматриваемой композиции, программа выводит следующее сообщение, подтверждая правильность композиции.

>> Validation\_n\_Queens\_Problem\_Solution

Input file name: kQueens\_Test\_Composition.mat

The size of a chessboard = 100

Elapsed time is 0.000014 seconds.

Composition size = 47

Composition is ok!

1. *Комплектация «сложной» композиции*

Рассмотрим другой пример. Пусть, в результате генерации, получена композиция, которая имеет следующий вид: *Q(1:100) =*

0 91 10 0 29 39 59 77 26 2

95 20 94 58 64 22 83 76 34 4

37 0 51 25 0 38 0 50 68 0

0 87 19 21 92 8 96 75 48 0

60 93 46 100 24 66 23 9 53 80

65 0 85 45 28 61 31 89 18 54

0 3 0 0 27 32 0 13 15 0

82 0 70 1 98 47 73 0 62 49

79 36 41 74 0 17 55 90 0 78

0 44 11 16 14 56 0 7 12 99

Здесь композиция состоит из *80* ферзей. Проверка показывает, что композиция правильная. При запуске программы «*Solution\_n\_Queens\_Completion\_Problem*», расчеты проводятся чуть дольше, чем обычно, но в результате программа выводит следующее сообщение:

>> Solution \_nQueens\_Completion\_Problem

Input file name: kQueens\_Test\_Composition.mat

The size of a chessboard = 100

Composition Size = 80

Number of free Positions = 20

Elapsed time is 2.187818 seconds.

falseNegSimCount = 5

Total number of usage the Back Tracking procedure = 5474

Columns 1 through 10

30 91 10 72 29 39 59 77 26 2

95 20 94 58 64 22 83 76 34 4

37 69 51 25 5 38 81 50 68 43

84 87 19 21 92 8 96 75 48 40

60 93 46 100 24 66 23 9 53 80

65 97 85 45 28 61 31 89 18 54

71 3 33 57 27 32 88 13 15 52

82 67 70 1 98 47 73 42 62 49

79 36 41 74 63 17 55 90 6 78

35 44 11 16 14 56 86 7 12 99

Solution saved in file: nQueens\_Test\_Completion\_Solution.mat

Как видим, алгоритм пять раз пытался с самого начала комплектовать данную композицию (falseNegSimCount = 5), и только с шестой попытки удалось это сделать. За это время более *5000* раз была выполнена процедура *Back Tracking*.

Сложность данной композиции состоит в том, что *80* ферзей расположены в ней таким образом, что в оставшихся свободных строках закрыты не только *80* столбцов, но закрыты и большая часть из оставшихся свободных позиций. Это результат работы диагональных ограничений, которые сформированы ранее установленными ферзями. При комплектаци, на последних шагах возникает такая ситуация, когда в двух или трех свободных строках остается по одной свободной позиции и попытка расположить ферзь в любой из них, сразу закрывет свободную позицию в другой строке (за счет диагональных ограничений). Это достаточно редкий и интересный экземпляр.

Такие композиции, которые не могут быть комплектованы, мы называем отрицательными. Ниже приведен пример отрицательной композиции: *Q (1:100)=*

21 31 89 20 50 46 28 39 78 66

57 45 37 52 41 100 92 8 29 19

58 75 6 25 96 34 71 95 15 23

79 73 68 88 67 72 35 98 86 1

83 12 24 44 32 61 14 70 99 80

60 47 81 77 7 59 2 38 0 13

17 51 90 55 49 16 22 26 11 0

76 18 91 87 63 93 9 0 64 94

4 0 56 0 10 5 30 85 65 97

42 40 0 69 54 0 0 62 27 3

После запуска программы комплектации, выводится следующее сообщение:

>> Solution \_nQueens\_Completion\_Problem

Input file name: kQueens\_Test\_Composition.mat

The size of a chessboard = 100

Composition Size = 92

Number of free Positions = 8

Elapsed time is 0.148292 seconds.

falseNegSimCount = 10

Total number of usage the Back Tracking procedure = 10084

This composition cannot be completied!

The error of such conclusion is less than 0.00001

Как видим, алгоритм пытался десять, раз с самого начала, комплетовать данную композицию. При этом, в совокупности, более *10000* раз была использована процедура *Back Tracking* для формирования ветви поиска решения. Но все эти попытки были безуспешными. Поэтому выводится решение о том, что с вероятностью *0.00001* данная композиция не может быть комплектована.

Если вывести число свободных позиций в оставшихся строках, то каждый раз результат будет примерно следующим:

*шаг-93:* 1 1 1 1 1 2 3

*шаг-94:* 0 1 1 1 1 1

или

*шаг-94:* 0 0 1 1 2 3

Или варианты близкие к этому. Какую бы свободную строку мы не выбирали для расположения ферзя, он закрывет последнюю свободную позицию в одной из оставшихся свободных строк.

1. *Пример комплектации композиции большого размера*

Проведем генерацию композиции для шахматной доски, размер которой равен *100\*106*. Пусть размер композиции равен *7 888 826*. Приведем первые *50* значений массива значений позиций ферзей:

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 9002222 0

0 0 0 0 0 0 0 13573604 0 0

84739423 0 0 0 0 0 39546789 0 0 0

21295044 99655382 0 0 0 0 0 0 0 0

- Время, которое необходимо для генерации такой композиции составило *17.59* секунд (все расчеты проводились на *DeskTop-13*, конфигурация этого компьютера описана в публикации в arhiv.org:<https://arxiv.org/abs/1912.05935> ) .

- Время, которое потребовалось для комплектации полученной композиции составило *384.51* секунд. Приведем первые *50* последовательных значений 1-мерного массива решения:

89582163 60368706 59234218 20261866 86935797 30224854 77724126 36762224 45267928 73444501

30441569 40452963 29233166 74466403 31027892 16431542 83945356 25649731 9002222 26312393

99907709 4403090 19464886 43088993 92254324 37208851 91076429 13573604 48390687 50900879

84739423 22553311 18099114 61465930 76001929 56522925 39546789 31942742 33849160 75874627

21295044 99655382 79211814 74172179 90416939 47905163 37217953 24667656 80381835 35026191

- Время проверки правильности полученного решения составило *15.71* секунд.

>> Validation\_n\_Queens\_Problem\_Solution

Input file name: kQueens\_Test\_Composition.mat

The size of a chessboard = 100000000

Elapsed time is 15.712291 seconds.

Solutions is ok!

1. *О линейности работы алгоритма комплектации композиций*

Во всех программах ведется учет времени работы алгоритма и по завершении выводится соответствующий результат. Допустим, что было проведено большое число экспериментов с различными композициями (как положительных, так и отрицательных) для различных значений *n*. Определим среднее время комплектации каждой композиции для каждого значения *n*. В результате получим выборку средних значений времени комплектации для каждого значения *n*. Если разделить среднее значение времени комплектации на соответствующее значение *n*, то получим «приведенное время счета». Это среднее время, которое затрачивает алгоритм для расположения ферзя в одной строке. Если рассматриваемый алгоритм является линейным по времени, то с ростом значения *n*, приведенное время не должно изменяться. В пределах небольшой погрешности приведенное время должно остаться постоянной величиной.

Для быстрой оценки алгоритма достаточно сформировать *20-30* композиций для списка значений *n=(100, 1000, 10 000, 100 000, 1 000 000, 10 000 000)*. Можно получить более точные результаты, если формировать и тестировать очень большие выборки композиций для различных значений *n*. В процессе исследования, в основном формировались выборки размером *100 000* композиций для различных значений *n*. Там, где суммарное время счета значительно увеличивалось, соответственно уменьшался размер выборки. Для *n=1000*, для исследования, была сформирована выборка, состоящая из одного миллиона композиций.

В исследовании рассматривался интервал значений размера шахматной доски от *7 до ста миллионов*. Для значений *n= (7,…,99)* в алгоритме отключаются некоторые процедуры. Это связано с тем, что для данного интервала значений *n* необходимо более тщательно выбирать индексы свободных строк и индексы свободных позиций в отобранных строках. Образно выражаясь, можно сказать, что данная зона является достаточно *турбулентной* и алгоритм расчета на этом участке (который составляет *0.000001* часть всего интервала) отличается по эффективности от основного алгоритма.