Contents

How to search Torus N Queens solutions

Matthias R. Engelhardt

January 2019

1 Introduction

The solutions to the Torus n queens problem are known and counted up to board size $n = 31$. As there are no solutions if n is divisible by 2 or 3, the next size is 35. There is a conjecture that the number of solutions is about $9.66 \cdot 10^{15}$. The challenge is to find the exact number, and to develop a feasible way to get this number.

In the following, we descibe our way to search and count the solutions for $n = 35$; the set of these solutions is called T_{35}.

2 Idea

2.1 The pure idea

There are two tools for the search: skeletons and the group XSim. All terms and objects will be explained in the following text.

A skeleton is a subset of the queens on the board, in a solution. The subset has at least 2 queens but may also contain all queens. Both extremal values are rare cases. For every solution, there is a unique, well defined skeleton.

Group XSim is used to exploit symmetry as much as possible. It has 235200 elements and is the biggest group we know for this purpose.

Group XSim acts both on the set of solutions and the set of skeletons, and it does it in a compatible way: if an element α transforms solution a to b, then it transforms also the skeleton of a to the skeleton of b.

This allows to use the 'Easy counting lemma' [1]. All we must do is:

- Build a table (or database) of all possible skeletons, reduced to a transversal for the action of XSim.
- For every skeleton in the table, count how many ways it allows to complete the skeleton to a full solution.
- For every skeleton in the table, compute the symmetry factor, multiply it by the number of completions, and sum up.

This is very rough; on the way to skeletons, we need also necklaces. A bit refined, the search and count consists of the following steps :

- **Necklaces:** Find all necklaces for size 35.
- **Skeletons:** Construct skeletons for size 35, and organize a transversal of all, under action of group XSim.
- **Database:** Set up a database which contains, per skeleton, a symmetry factor and the number of solutions found for it; for the start, the number of solutions is initialized with 0. An entry may also contain trace information, for instance the time needed for the search for this skeleton.
- **Complete:** for all skeletons in the database: fill up with queens, to get the number of solutions for the skeleton. Fill it into the database.
- **Sum up:** Scan the database and sum up the products of symmetry factor and number of solutions.

Main effort is the search step; it can be parallized. This is important for the feasibility. The task which should be done in this step is not so complex; this allows to migrate a part of it to FPGA devices, like it was done for the (non-torus) search for Q_{27}[2].

Compared to the search for T_{31} in 2003, there are two important improvements: use of the EasyCounting lemma, and use of group XSim which means similarity plus a sort of 45 degree rotation. These improvements are discussed in the next sections; after that, the complete algorithm is described.

2.2 Practical drawbacks

Main problem: there are too many skeletons. But a large portion of skeletons allows no completion; we can neglect them. We have still the problem to find out which skeletons we can neglect.

The current attempt to overcome this problem uses the fact that skeletons can be ordered by two properties which are well defined: their maximal necklace, and the number of queens contained.

The extreme growth of the number of skeletons happens at two places: for very small necklaces, and for high necklaces. In the case of T35, small necklaces means some necklaces of 2 or 3 beads, and high means necklaces of 12 or more beads. The case of the one bead necklace is special, but can be handled in the same way.

For these problem areas, we reverse the way of search: we search first the set of complete solutions, just as it was done in the T31 search; then we extract their skeletons and sort the solutions by their skeleton. These skeletons are then included in the database. I call this a 'cut', meaning that we cut off the search for all skeletons.

Another small point is that the computation of the symmetry factor for a skeleton takes some CPU which gives an effect if it is made for very many skeletons. The easy measure to take is to compute it only if we know that there are solutions for this skeleton.

An open point is how to test the result; for T31 and lower, this was done by compare of a random sample of regions where I compared the new result with the numbers computed by a much simpler program, for the region. But this needs that a transversal of all solution is saved on disc or some other medium; for T35, we try to avoid to save all the solutions.

3 Easy counting lemma

The Easy counting lemma allows to count solutions without spending much effort to eliminate duplicates. It is discussed in more detail in a separate paper[1].

The main precondition for the use of the Easy counting lemma is a mapping of the big set T_{35} to a smaller set S; this mapping must be compatible with the group action. The idea is to use the set of *skeletons* or skeleton candidates as smaller set.

Before we came to skeletons, we tried with lines which are populated in a maximal way. So we must explain what we mean with populated lines, in the context of chess board and queens problem: There are many types of lines; we mean straight lines, i.e. images of the set $\{(k, 0, 1) | k \in \mathbb{Z}_n\}$ [1] under some affine mapping. Rows and columns belong to these straight lines, and diagonals as well. But they are all not of interest in the torus queens problem context: due to the torus queens rules, each of them is populated by exactly one queen. Each of them is 'maximal populated', but that does not help.

The next sort of lines are knight lines like $\{(k, 2k, 1) | k \in \mathbb{Z}_n\}$. They allow for distinction of solutions: the population may vary on them, from 0 to n. Of course, there must exist lines with maximal population. The population of queens on a line allows more distinction, not only the number of queens, but also their placement (ordering) on the line. But this should not change under the movements discussed in the next section. This leads to use of the mathematical notion of a necklace: for every line on the chessboard, we take the sequence of cells, and assign a white bead to the empty cells, and a black one if there is a queen in the cell.

A definition of a mathematical necklace is found in the English Wikipedia[3]. The base idea is to handle the line as a closed loop; sequences of beads are considered equivalent if the loop is rotated. The article in Wikipedia suggest the term Bracelet if also turning over the loop is considered equivalent. We use an even larger group, and allow to wrap up the loop k times, for all k with $\gcd(n, k) = 1$. Instead of inventing one more term for the resulting objects, we call all of them necklaces, and would speak of necklaces under rotation if we mean the necklaces of Wikipedia, necklaces under the dihedral group for the bracelets of the Wikipedia article, necklaces under similarity or wrap necklaces

[1]We use the notation $\{(x, y, 1) | x, y \in \mathbb{Z}_n\}$ for the affine plane; it is 2-dimensional, as the third coordintae is always 1.

for our enlarged use. In the following text, necklace means always similarity necklaces.

We have defined an order for necklaces also; using it, we can select lines containing a maximal necklace of queens. Necklaces having more black beans are considered higher, and if the number of black beans is equal, the necklace with black beans in lower position is lower. But keep in mind we take always the lowest value within an orbit of the similarity group.

But there is no unique line which has this maximal necklace of queens: several lines can be populated by the same maximal necklace of queens. To overcome this problem, we define a skeleton of a solution as the subset of all queens belonging to a maximal necklace. Again somthing to define: a *skeleton* is a subsets of the squares of the $n \times n$-board which is admissible under the torus n-queens restrictions and fulfills the condition that every square belongs to a maximal knight line.[2]

This gives a well defined mapping from solutions in T_n to skeletons. In general, it is neither injective nor surjective. There are many skeletons which cannot be completed to a solution, and many skeletons have different possibilities of extension to a torus solution.

4 The group XSim

Group XSim is a subgroup of the group of affine bijective movements of the two-dimensional plane over the ring of integers modulo n, $\mathbb{Z}_n \times \mathbb{Z}_n$. It is defined for odd n only. It is generated by translations, rotations, reflections, extensions; for extensions, the extension factor x must be relative prime to n, i.e. $\gcd(x, n) = 1$. They can be described by the matrices $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$(horizontal shift), $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ (vertical shift), $\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$(horizontal reflection), $\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$(vertical reflection), $\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$(90 degree rotation), $\begin{pmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 1 \end{pmatrix}$(extension by k), and finally the 45 degree rotation $\begin{pmatrix} 1 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$[3].

The 90 degree rotation may be omitted here; but note that the 45 degree rotation is a bit special: executes twice, it does not give the 90 degree rotation; it gives the 90 degree rotation, concatenated with extension by 2, instead. The extension by 2 may be compensated by an extension by $(n+1)/2$ which is an integer as n is odd.

[2]In a previous paper, this was the definition for a skeleton candidate, and skeleton was used only if it was the skeleton of some torus solution; the new definition is simpler.

[3]We have chosen these matrices as generators because they are simple and easy to keep in mind; there are smaller sets of generators. At present, we do not know minimal sets of generators or sets of generators of minimal size.

Note that the 45 degree rotation maps rows and columns to diagonals, and vice versa. As all of these lines contain exactly one queen, the image of a solution is again a solution.

Due to the 45 degree rotation, the use of knight lines must be extended: normal knight lines like $\{(k,2k,1)|k \in \mathbb{Z}_n\}$ are mapped to lines like $\{(k,3k,1)|k \in \mathbb{Z}_n\}$ which we call knight-3 lines. The knight-3 lines are mapped back to normal knight lines; if we take the maximal population over both normal knight lines and knight-3 lines, everything works fine, we need no further extension.

5 More things to keep in mind

For use of the Easy counting lemma, we start with searching all skeletons. A skeleton has a unique maximal necklace, and necklaces are the start point for the search of all skeleton candidates.

Skeletons are also sets for an action of group XSim of the last section, as are all subsets of $\mathbb{Z}_n \times \mathbb{Z}_n$. In our algorithm, we do not need all skeletons: we need only a transversal of them, under the group action.

5.1 Modifications to the search algorithm, start from database

If we have a skeleton, we try to fill up with queens, until complete solutions are reached. To do it, we use the usual n queens torus search algorithm; but we need a small modification: neither should the search result in a knight line with a higher necklace of queens, nor should there be additional queens on maximal populated knight lines. The first case would mean that the maximal necklace has changed, the second, that the skeleton has grown, even if the maximal necklace is still the same. The solution maps no longer to the skeleton of our start point, which violates the precondition of the Easy Counting lemma.

5.2 Algorithm of the search for skeletons

First: we do not search all skeletons, we search a transversal of them, instead. Transversal means transversal under the action of group XSim.

The search for all skeletons is organised by necklaces, in the first place. This is quite natural, there is nothing more to explain.

For a given necklace, the main help tool in the search is a table; it contains all possible positions of the necklace under the action of XSim. The algorithm starts with placing one necklace of queens in the simplest position on the board, and write it out as the first skeleton. This is the start of an iteration: the file of only one skeleton becomes the base file of the first iteration.

Every iteration tries to combine the skeletons in the base file with all skeletons in the table. Combination means: first find the set difference of the queens, or in other words: the set of queens from the table position which are not contained in the skeleton from the base file. Then check, that these queens have their own free rows, columns and diagonals, as n queens rules must be fulfilled. In the third place check, that they will not lead to a higher number of queens or a higher necklace on any of the knight diagonals. If all is OK, the set union of the queens is a new skeleton. It is normalised to its lowest form, under action of XSim. Then it is stored in a table or file of additional skeletons. This table or file is organised by the number of contained queens. Note that the new skeleton may already be there, stemming from a different combination of skeleton from base file and position from the help table. In this case, nothing happens: one instance in the file is sufficient. When all combinations are built, the file of the additional skeletons having least queens per skeleton is written out and becomes the base file for the next iteration.

The iteration will stop when all skeletons are found. This is sufficient for theory; in pratice, the sets of skeletons may become very large and the search time very long. For these cases, the algorithm has also a limiting parameter; if the number of additional skeletons in the above algorithm exceeds this number *at the end of an iteration*, the handling terminates. It writes out the additional skeletons into a separate file called 'cut file', for the given necklace; at this place, it is important that the file contains all additional skeletons, not only those of the next size which would become the next base file if the iteration were continued. The next subsection describes how to continue with this cut file.

5.3 Cut handling

It turns out that the search for skeletons leads to high numbers, in special situations; most likely, almost all of these skeletons cannot be extended to complete solutions.

For all 'cut' situations, the search sequence is reversed: we search first complete solutions, and derive the set of interesting skeletons from it. For the search, the algorithm starts with some prepositioned queens, but it does not contain the extra restrictions of Sec. 5.1.

As already mentioned in Sec. 2.2, high numbers of skeletons occur both for small necklaces and high necklaces. Small necklaces means necklaces of only 2 or 3 beads, at least for T19 up to T31. Possibly, there are also cases of 4 beads, for T35. The small necklaces lead to skeletons consisting of many 'necklace lines', combined in different ways. On the high necklace side, the number of the necklaces grows very high, and each of them leads to at least one skeleton.

Before we come to the cut handling of small necklaces, note what is special for the first two: the first necklace (lowest in our order) contains only white beads; the only possible skeleton is the empty set, and this cannot be the skeleton of torus solution. The second necklace has only one black bead. It can be the maximal necklace of a torus solution, but in this case, the skeleton is already the complete solution. If the skeleton had less queens, there must be other queens for the completion, and their lines would also bear a maximal necklace which means they would belong to the skeleton - a contradiction.

In the program, there are 4 cut situations; three of them are almost the same, but I enumerate all 4 here:

Table 1: Pro and con for cut

Pro	Contra
Reduces skeleton search	Same solution found several times
avoids to inflate database	Solutions must be stored
	duplicates must be excluded
	skeletons must be extracted from the solution

- **cutMany:** there were too many skeletons, from a given necklace. In this case, the cut search starts with the file of all skeletons which were no more extended in the normal skeleton search.
- **cut1:** the necklace 1: one queen is set on 1/1, then search with restriction to necklace 1.
- **cutFull:** the program accepts a hint given with the parameters: set queens for the first 'necklace line', then search with restriction to the actual necklace.
- **cutHigh:** for every necklace of a given number of beads, one record is given in a file. the search starts with each of theses preplacements, without restriction for a necklace. This means that higher necklaces can be produced in the search.

The first three situations are essentially the same, but they arise in different places of the program.

The program accepts a parameter which tells the number when the program should stop generating more skeletons for a necklace, and undertake a cut run of type cutMany instead. A second parameter gives the number of beads where the cutHigh should start.

Pro and con for cut hanling is given in table 1.

I have an idea for the highCut situation, how to reduce the number how often the same solution is found; this is not yet implemented.

Bibliography

References

[1] M. R. Engelhardt, T. B. Preußer, An easy counting lemma, Discrete Applied Mathematics.
URL `https://doi.org/10.1016/j.dam.2018.10.001`

[2] T. B. Preußer, M. R. Engelhardt, Putting queens in carry chains, No. 27, Journal of Signal Processing Systems 88 (2016) 185 – 201.
URL `http://dx.doi.org/10.1007/s11265-016-1176-8`

[3] unknown, Necklace (combinatorics), Wikipedia article.
URL `https://en.wikipedia.org/wiki/Necklace_(combinatorics)`