Saint Nicholas, Rooks and Graph Matchings

Rianne de Heide

November 27, 2018

Saint Nicholas

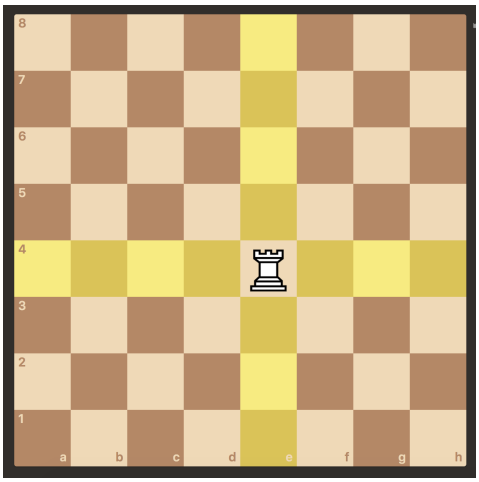
Last week students from the Coornhert Gymnasium in Gouda came up with the following problem. On the fifth of December the Dutch celebrate *Sinterklaas* (Saint Nicholas). Groups of friends or family gather around a pile of presents, creatively disguised and provided with a personal poem. Weeks before the party, everyone in the group is randomly assigned a person to whom they give a gift. Nowadays, there are plenty of websites for picking the names, and it is possible to enter constraints. The group of eleven friends from Gouda entered Table 1 into some website and person B got assigned person K to buy a present for. He also wondered: now I know my own assignment, and I know the constraints we drafted together, how many possibilities for assigning the remaining tickets are left? And the next question that pops up is: can we calculate this efficiently for a given table?

	A	B	C	D	E	F	G	H	I	J	K
A	x										x
B	x	x	x	x	x	x	x	x	x	x	0
C			x		x		x		x	x	x
D				x							x
E			x		x				x		x
F	x			x		x					x
G		x	x				x		x		x
H								x			x
I		x			x		x		x	x	x
J		x	x						x	x	x
K											x

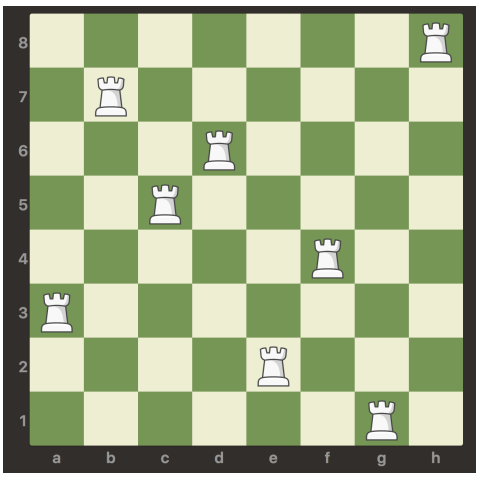
Table 1: Sinterklaas table with constraints

The brute force solution

Naturally we start with removing row B and column K from the table. Now we want write a computer program to simply enumerate the possibilities. One way to do this, is to look for the row with the highest number of constraints (breaking ties randomly). In the table above, we could start with row C. We then look at the number of possibilities when person C gets



(a) Movements of the rook



(b) Rook placing

Figure 1: Rooks

the ticket with A's name on it: we eliminate row C and column A, and we repeat the process: we look at the row with the highest number of constraints in the new 9×9 table we are left with, and so on. For the problem of Table 1, the computer provides us with the answer in about 15 seconds: there are 105866 possibilities.

Conjecture In this way — by working from rows with the highest number of constraints to the row with the lowest number of constraints — we do not run into impossible situations, by which I mean that we are left with a person that is not assigned a ticket, yet the constraints are such that it is not possible to do so anymore.

Rooks[1]

This problem reminded Nicolette of a problem given to first year mathematics students at Leiden University. How can we place rooks on a chess board such that they are not able to capture one another?

The rook is a chess piece that can move up-and-down and side-to-side (Figure 1a). We can place 8 rooks on a chess board in such a way that they cannot capture one another (Figure 1b).

Imagine that our Table 1 is just a larger (11×11) chess board, in which we want to place $11 = |\{A, \ldots, K\}|$ rooks, and we are not allowed to put rooks on the crosses. So every permutation (ticket assignment) that is allowed, is equivalent to a rook-placing. We now do the following: we cut out the fields in our 11×11 board that have a cross on them. What we get is a chess board with a weird shape, and we call this 'board C'. Now let $r_i(C)$ be the number of ways to place i rooks on board C, for all $i \in \mathbb{N}$ (that is, the natural numbers: $1, 2, 3, \ldots$). The number of allowed permutations on an $n \times n$ board is $n!$ (which is the number

[1]Many thanks to Nicolette van Splunder for bringing this problem of rook placements to my attention.

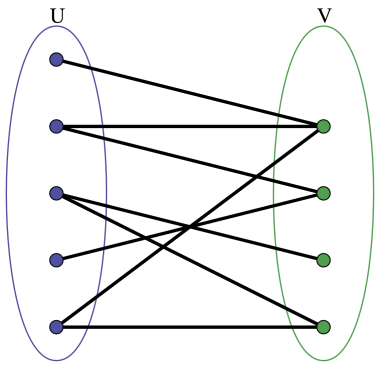


Figure 2: Bipartite graph (figure from wikipedia)

of placements when there wouldn't be any constraints), minus the number of placements in which at least one rook sits on a forbidden field, plus the number of placements in which at least two rooks sit on a forbidden field, ..., plus $(-1)^n$ times the number of placements in which all rooks sit on board C. In math: $n! - (n-1)r_1(C) + (n-2)r_2(C) - \ldots + (-1)^n r_n(C)$. We can form the 'rook polynomial':

$$R(x, C) = r_0(C) + r_1(C)x + r_2(C)x^2 + \ldots + r_m(C)x^m.$$

How can we calculate $R(x, C)$ for our Table 1? We can subdivide board C into smaller pieces, and we can make use of the following rules:

1. When C consists of two parts, A and B, such that no field of A is in the same row or column of B, then we have: $R(x, C) = R(x, A)R(x, B)$.

2. Let f be a field of C, and let D be the board we obtain when we remove every field from C that is in the same row or column as f. Let E be the board we obtain by removing field f from C. Then we have: $R(x, C) = xR(x, D) + R(x, E)$.

Implementation for Table 1 yields the same answer as before: there are 105866 ways to place rooks in Table 1 such that no rook is on a forbidden field, and no rook can capture another rook.

Perfect matchings of bipartite graphs

The question remains: how hard is it to compute the number of allowed permutations for a given Sinterklaas table? It turns out that we can match this problem to other problems of which (we think) we know how hard they are. The first one is finding all perfect matchings of a bipartite graph. A *graph* is a mathematical object consisting of *vertices* (also called nodes, think of points) and *edges* (think of lines connecting some of the points). In Figure 2 we see a graph, it has 9 vertices and 8 edges. There is something special about the graph in Figure 2: it is called *bipartite*. That means that the vertices can be divided in two disjoint sets, called U and V in the figure, such that every edge runs from a vertex in U to a vertex in V.

Now imagine a bipartite graph just as in Figure 2, where the set U consists of vertices representing the names of the partiers and the set V consists of vertices representing the tickets. Finding a way to distribute the tickets over the partiers amounts to drawing an edge from every vertex in U to a vertex in V such that every vertex is incident to exactly one edge. That is called a *perfect matching*. How do we introduce the constraints corresponding to Table 1? Well, we first draw all edges that are allowed. So we draw an edge from A $\in U$ (person A on the left) to B $\in V$ (ticket B on the right), but we do not draw an edge from A $\in U$ to A $\in V$, because person A is not allowed to buy herself a present, as we can see in the table. If we are done, we have a graph with a lot of edges (but not all!) between the vertices in U on the left side and the vertices in V on the right side. We have vertices and edges, so we have a *graph*, and we see that it is also *bipartite*. If we enumerate all *perfect matchings* on this graph, we have found the solution to our problem. And, we have an answer to our second question: how hard is it to compute? It is thought that finding all perfect matchings of a bipartite graph is notoriously hard, and in *computational complexity theory* there is a ranking of hardness of problems. You might have heard of P and NP — problems belonging to the category P are easy to solve for a computer, problems of category NP are easy to check for a computer, but not easy to solve. After P and NP a lot of other letters follow, until we arrive at the complexity class called #P-complete (pronounced as *sharp-P-complete*). Some problems that live in #P-complete would be 'easy' (that is, complexity class P) if you would only want to know one solution. It is quite easy to find one perfect matching of the bipartite graph corresponding to Table 1. But if you want to count how many there are, it becomes very, very hard. That means that there is not really a smarter/faster way of computing the possibilities of our problem than what we did above.

The matrix permanent

We can match our problem to something called computing the *permanent* of a matrix (thanks to Wouter Koolen for finding this). Let's take another look at Table 1. Now we put zero's on the crosses, and ones on the fields that are open. We obtained a matrix with zero's and ones on the entries. The number of ways to assign tickets to persons or to find a perfect matching of the a bipartite graph corresponds to computing the *permanent* of this matrix (let's call it A), which is defined as:

$$\mathrm{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

The sum is over all permutations of the numbers $1, 2, \ldots, n$, and $a_{i,j}$ stands for the entry of the matrix A in column i and row (j). In case you want to look it up, S_n in this formula is the finite symmetric group — I think it's safe to assume we want to celebrate Sinterklaas with a finite number of people. And, indeed, the computation of the permanent of a $(0, 1)$-matrix is #P complete. That means that if we want to celebrate Sinterklaas with 15 people, we have some time to do other things while the computer is computing the possibilities, but if we want to know the answer for celebrating Sinterklaas with 100 people (by the way, that is ridiculous, it will take days to read all the poems and open all the presents), we better start computing (rough guess) ages in advance. Of course your computer would crash way before that.