Charles University Prague

Constraint Programming

Christoph Saffer

March 28, 2017

Peaceable Armies Of Queens

Documentation

Contents

1 Description Of The Problem

We consider a chess board with size D and queens from both colors, black and white. We want to place all queens on the given board such that no queen can beat a queen from another color. So we have two armies of queens where no queen can reach a queen from the other team within one round. The queens' movement are determined by the common chess rules. They are allowed to move any number of steps in the same row, column and diagonal in each direction. The goal is to maximize the number N of queens (N black queens, N white queens) for a given size D.

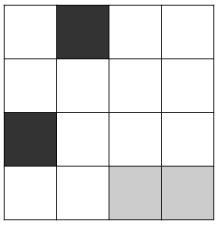


Figure 1: Example for D=4

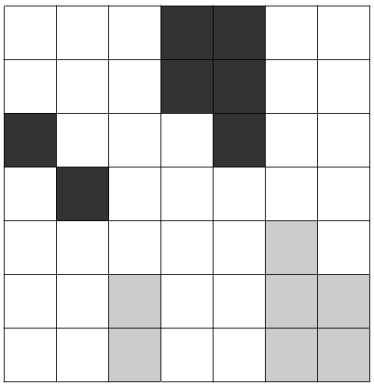


Figure 2: Example for D=7

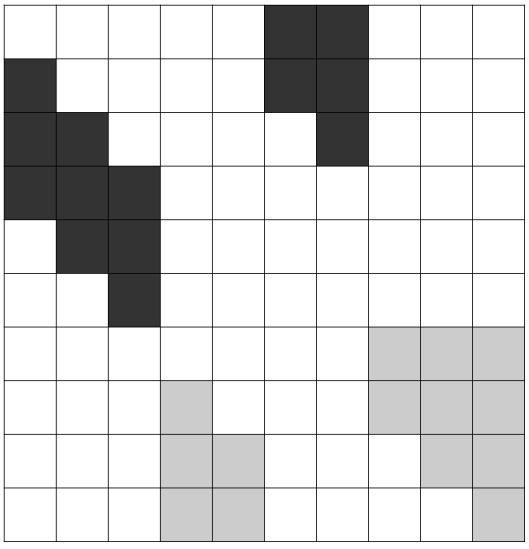


Figure 3: Example for D=10

2 Constraint Model

We use a constraint model to calculate the maximum number of possible queens for a given size D. There we instantiate D^2 variables, one for each field on the board. Furthermore, we encode the black queens with '-1', the white queens with '1' and the free fields with '0'. In the end we expect an assignment of each variable in the domain $\{-1, 0, 1\}$ considering the following constraints.

- the same number of "-1" and "1" assigend to the variables
- N variables assigned with "-1" and N variables assigend with "1"
- each row contains fields only with "-1" and "0" or "1" and "0"
- each column contains fields only with "-1" and "0" or "1" and "0"
- each diagonal constains fields only with "-1" and "0" or "1" and "0"

Basically the constraints claim an assignment with the same number of black and white queens and with no different queens in the same row, column and diagonal. Finally, we will maximize the number N of queens for a given size D.

0	-1	0	-1	0	-1	0	-1	0
-1	0	0	0	0	0	0	0	0
0	-1	0	-1	0	-1	0	-1	0
-1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0	1
-1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0	1
-1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0	1

Figure 4: Model for D=9

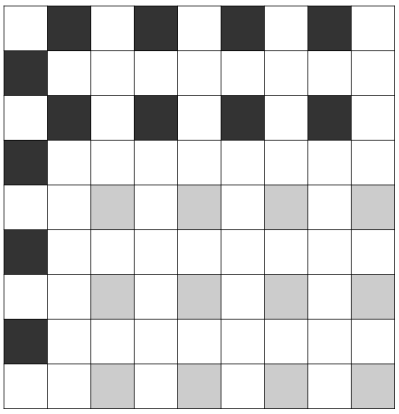


Figure 5: Example for D=9

3 Prolog User Manual

3.1 Including the files

First of all, we are working with Sicstus Prolog 4.3.5. To use the implementation we include the libraries 'clfpd' and 'lists':

```
| ?- use_module(library(clpfd)), use_module(library(lists)).
```

Then we have to consult the prolog file to use all the procedures:

```
| ?- consult('/PATH/TO/FILE/queensarmies.pl').
```

Now we can ask for the maximum number of queens for a size D with the procedures qa0(S, D, N), where S is a matrix containing $\{-1, 0, 1\}$ representing the chess board, D the size of the chess board and N the number of queens. If we do not assign a value to N, it maximizes the number of queens for the given size D. Example with size 5:

```
| ?- qa0(S,5,N).
S = [[0,-1,0,-1,0],[-1,0,0,0,0],[0,0,1,0,1],[-1,0,0,0,0],[0,0,1,0,1]],
N = 4 ?
yes
```

We also can ask for a specific N, even if it is not the maximum for the given size. It will answer if it is possible or not:

```
| ?- qa0(S,5,3).
S = [[0,0,-1,0,0],[0,0,-1,0,0],[-1,0,0,0,0],[0,0,0,0,0],[0,1,0,1,1]] ?
yes
```

3.2 Implementation of the constraints

To implement the constraints we can use several built-in predicates. To ensure we have N black queens and N white queens, we use global_cardinality. It checks the number of occurrences of a given value in a list:

```
global_cardinality(Sol1, [-1-N, 1-N, 0-_]),
```

We also use global_cardinality for checking the consistency of each row, column and diagonal. We sort all of them in lists (so we have list of lists) and write an extended global_cardinality predicate called check_rows_a to check their consistency.

```
check_rows_a([]).
check_rows_a([H | T]) :-
global_cardinality(H, [1-N1,0-_,-1-N2]), (N1#=0 #\/ N2#=0),
check_rows_a(T).
```

3.3 Maximazation of N

In order to maximize the number N of queens, we determine a domain for N and include a maximisation refered to N in the labelling of the variables. Obviously, $L = D^2$ is the total number of fields we have on the chessboard. So the domain of N is $\{1, \lfloor \frac{D^2}{4} \rfloor\}$, because it is not possible to assign half of the fields with '-1' and '1' without violating any constraints. The labelling is done as follows:

```
N#> 0, N#< L/4,
labeling([maximize(N)],Sol1).
```

In the first tests, the standard labelling option 'leftmost' where the leftmost variable is selected next for assignment is used. After some exploring, we can figure out that the option 'ffc' where the variable with most constraints and smallest domain is assigned next. The results are shown below.

```
N#> 0, N#< L/4,
labeling([maximize(N), ffc],Sol1).
```

3.4 Optimatization of the program

In order to optimize the runtime of the program we can try to cut the domains of some variables in the beginning. It is clear that we can assume that the first row contains only '-1' and '0', thus we already have some symmetry breaking what decreases the runtime. This is done in the procedure qa1:

```
row(A, 1, Row1),
domain(Row1, -1, 0),
```

After running some tests, for an optimal solution it seems that not also the first row but also the first column that only contain fields with '-1' and '0'. At the same time the bottom right field is always assigned with '1'. This brings us to some cuts for the domains, which improves the runtime. Notice that we have no proof that these cuts are not leading to some loss of possible solutions, but at least it helps for getting a lower bound for bigger sizes of the chessboard where the runtime of the previous procedures is too high. These extra lines are included in the procedure qa1_extended:

```
column(A, 1, Col1), domain(Col1, -1, 0),
row(A, D, RowD), column(A, D, ColD), diagos(A, Diag, 0, D),
domain(RowD, 0, 1), domain(ColD, 0, 1), domain(Diag, 0, 1),
nth1(D, RowD, LastElLastRow), LastElLastRow is 1,
```

Furthermore, we can try to cut the domain of N to deny some obvious checks. It is clear that the maximum number of queens does not decrease when we increase the size D of the chessboard. Thus, we can change the lower bound to the maximum number of queens of the size D-1. This means we call the same procedure with size D-1 to get a lower bound of N of the current size D. We get a recursive procedure which goes until $D = 2$ and outputs in each recursion the minimum number of queens for the next larger D. These extra lines are included in the procedure qa1_rec:

```
D1 is D-1,
qa(A1,D1,N1),
...
N#>= N1, N#< L/4,
```

4 Tests

4.1 Results

In the following we can see the runtime in seconds of the different procedures qa0, qa1, qa1_extended and qa1_rec for different sizes D of the chessboard. The tests were processed with an Intel Core i5-6200U CPU 2.30GHz x 4 on Windows 10. In the first table the results from the standard labelling option 'leftmost' are shown:

Size D	Queens N	qa0	qa1	qa1_extended	qa1_rec
1	0	0.0	0.0	0.0	0.0
2	0	0.0	0.0	0.0	0.0
3	1	0.0	0.0	0.0	0.0
4	2	0.015	0.015	0.0	0.015
5	4	0.11	0.078	0.016	0.078
6	5	2.985	1.687	0.156	1.734
7	7	59,5	33.25	2.5	34.375
8	9	1795	990.9	68.812	985.3
9	12	-	-	1387	-
10	14	-	-	-	-

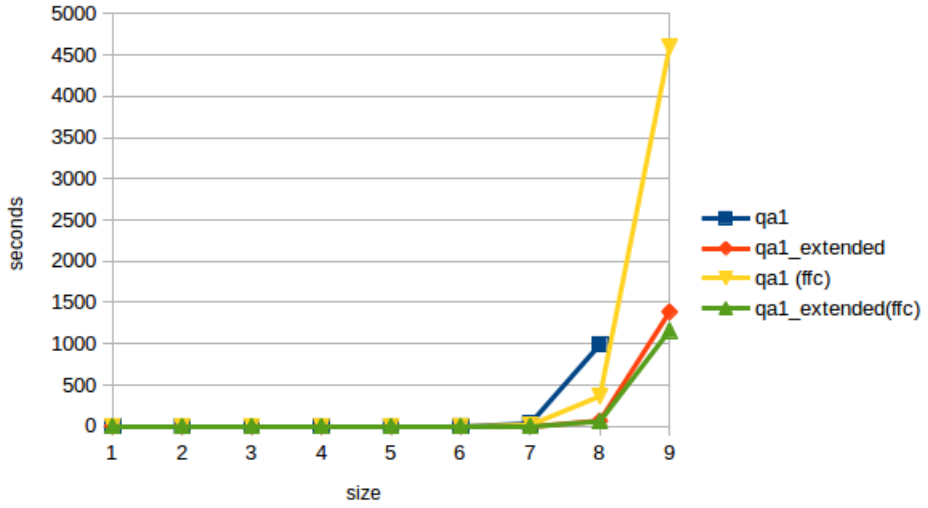
Here, we can see the results from the labelling option 'ffc':

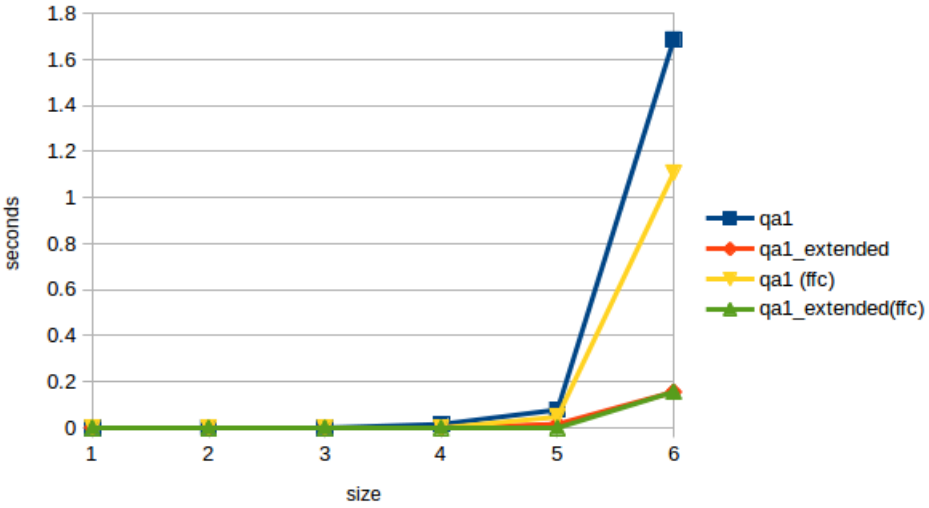
Size D	Queens N	qa0	qa1	qa1_extended	qa1_rec
1	0	0.0	0.0	0.0	0.0
2	0	0.0	0.0	0.0	0.0
3	1	0.0	0.0	0.0	0.0
4	2	0.015	0.0	0.0	0.0
5	4	0.078	0.047	0.0	0.062
6	5	1.813	1.109	0.156	1.172
7	7	27.41	16.13	2.359	17.3
8	9	540.4	364.1	60.2	395.1
9	12	-	4599	1160	5023
10	14	-	-	-	-

4.2 Interpretation

As we can see in the tables, qa1_extended is the fastest procedure.
In contrast to qa0, qa1 does not check the same symmetrical assignments more times, because the first row on the chessboard is already determined with black queens or free fields. Furthermore, there are fewer possibilities for feasible assignments what is very valuable for the runtime. The improvement in qa1_extended is even more forcing this thought - denies more symmetrical assignments from checking and decreases the number of possible assignments. Therefore, this is even faster than qa1, but we can not be sure if some solutions are cut from the domain. Thus, the results from qa1_extended can be only seen as lower bounds for a solution, but in all tests where we have a solution in reasonable time, the solution of qa1 and qa1_extended is equal, but it does not say anything about bigger chessboards.
The idea of qa1_rec of checking the solution for the size D-1 first to get a lower bound for N does not work in practice very well for small sizes, because we lose most of the time in

checking if it is not possible to place N queens for a given size D. It might be helpful for bigger chessboards, but it is difficult to test.
As an interesting fact we can notice that the change of the labelling decreases the runtime of all procedures approximately by half except for qa1_extended where the effect is quite small. In the following charts this effect is shown by comparing the procedures qa1, qa1_extended and their versions with labelling option 'ffc'. The second chart shows the details from the first chart until size 6:





Finally, we can conclude that we have quite fast algorithms but the runtime increases really quick the bigger sizes we get which is illustrated in the charts. We can try to exclude not possible assignments from the beginning or deny symmetry breaking at some points, but sooner or later the complexity of the problem is too high for this constraint model to get a solution for bigger sized chessboards.