The Execution Time of N Figures Problem Cannot Be Polynomial

Written by Jozsef Kiss (KissCode Systems Kft, Hungary)
Site: https://openso.kisscodesystems.com/
Date: 01.01.2018.

0. ABSTRACT

There is an existing and non-empty subset of algorithms in which the elements can resolve the N figures (N queens and its extensions) problem. These implementations have the property that the Running time of them will be increased dramatically as the size of the chessboard (and the number of the put figures) increases.
The current world record is known as 27 using queens so the possible valid positions (no queens attack each others in these every placings) are known and counted in this size of the chessboard.
There are many attempts to reduce the Running time of that implementations (using bitwise operations, using symmetry, using precalculated attacking map and only valid positions, etc.) but the tendency cannot be changed considerably. If this could be possible then all of the possible solutions could be known on the chessboard for example in size 100, but we cannot know this information. We would like to show that it is not possible to calculate the valid positions of the N chess figures and to count them quickly or in Polynomial running time.
The P!=NP suspicion can be confuted by solving the N queens problem quickly, anyway, which is one of the millennium problems.

1. DEFINITIONS

1.1 Dimension (D)
the count of all of the used figures to use
1.2 Chess board (C)
0 .. (D ^ 2 - 1) ordered sequence
for example: the representation of the C4x4:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

1.3 Good position
this property is declared between any 2 elements of C: Ci and Cj (i,j contained by N and i,j <= D ^ 2 - 1) and means that that 2 elements are not under attack by each other
1.4 Good placing
elements in count D of C Chess board in which every 2 elements are Good position
1.5 N chess figures problem
to create every Good placings of chess figures in count D on the C chess board in size D and count them
1.6 N chess figures algorithm (A)
algorithm that can place the chess figures in all of Good placings in an optimized way so there is no part of this algorithm that doesn't work on the N chess figures problem
1.7 Verifying time (T0)
the time while the computer can verify any 2 elements of C: they are in Good position
it doesn't depend on the current time and nor to the 2 elements of C considered as tiny constant
1.8 Counting time (T1)
the time while the computer can count a Good placing
also considered as tiny constant as T0
1.9 Running time (TR)
the full time duration while the implementation of A can start and can end its running so the difference between the end and start times
TR only depends on the number of the CPU commands to execute
it depends on the value of D indirectly
1.10 Polynomial running time
there is a k positive integer as the Running time of implementation of A is O (D ^ k) to any value of D

2. ANALYSIS

2.1 Assumption: it is possible to to solve the N chess figures problem by running the implementation of A in Polynomial running time.

2.2 The basic expectations of the N figure problem to the A are:
- be able to determine the property of Good position from all elements of any subsequence of C
- be able to count the Good placings

2.3 Let the A algorithm to do exactly the aboves and no more.
Thus, A is an algorithm that can create all of the subsequences of the Good placings within exactly 0 time and gives only Good placings.
Furthermore, A will be able to decide that the subsequences are really Good placings and will be able to count the Good placings.

2.4 The algorithm will do the following jobs:

A can decide the Good placing condition of elements counted as D under

(1) $0 + 1 + 2 + .. + (D - 1) = 1/2 * D * (D - 1)$

calculations.
This can happen within

(2) $T0 * 1/2 * D * (D - 1)$

time according to (1).
This new Good position has to be added (found ++) to the count of already found solutions which happens within T1.
Let it be

(3.a) $G = G (D)$

the count of the Good solutions belonging to D Dimension.
We cannot know anything about this but on large D values

(3.b) $G (D) < G (D + 1)$

Then all of the time needed by the calculation of all of the aboves according to (1), (2) and (3.a):

(4) $TR (D) = T0 * 1/2 * D * (D - 1) * G + T1 * G$

We will use large D values so

(5) $D * (D - 1) \sim D \wedge 2$

Thus, the (4) expression will be the following using large D values:

(6) $TR (D) = T0 * 1/2 * D \wedge 2 * G + T1 * G$

Let it be according to 1.10 and (6):

(7)
$$TR (D) = D \wedge k$$
$$T0 * 1/2 * D \wedge 2 * G + T1 * G = D \wedge k$$

so we will search for the constant k value to this Running time located on left side.

2.5 Let's see the values of k to the (7) of 2.4.

2.5.1 Let k = 1, then (7):

(8)
$$T0 * 1/2 * D \wedge 2 * G + T1 * G = D$$
$$1/2 * T0 * G * D \wedge 2 - D + T1 * G = 0$$

This is a 2nd equitation to D which has at least 1 solution when

(9)
$$(- 1) \wedge 2 - 4 * 1/2 * T0 * G * T1 * G >= 0$$
$$1 - 2 * T0 * T1 * G \wedge 2 >= 0$$

T0 and T1 are tiny constants but after (3.a), G depends on the size D of the Chess board. Then (9) cannot be true above a specific value of D according to (3.b).
This is the reason of why the (7) cannot be true to any D when k = 1.

2.5.2 Let k = 2, then (7):

```
  ( 10 )            T0 * 1/2 * D ^ 2 * G + T1 * G = D ^ 2
          ( 1/2 * T0 * G - 1 ) * D ^ 2 + T1 * G = 0
                    ( 1 - 1/2 * T0 * G ) * D ^ 2 = T1 * G
                                          D = sqrt (
                                           T1 * G / ( 1 - 1/2 * T0 * G )
                                                  )
```

This can be interpreted if

```
  ( 11 )    1 - 1/2 * T0 * G > 0
```

Similar to (9), it can be seen in (11) that it cannot always be true according to (3.b), so (7) cannot always be true to any D when k = 2.

2.5.3 Let's see (7) in general or unmodified case.

```
  ( 12 )                        T0 * 1/2 * D ^ 2 * G + T1 * G = D ^ k , k > 2
                      D ^ k - 1/2 * T0 * G * D ^ 2 - T1 * G = 0
          D ^ 2 * ( D ^ ( k - 2 ) - 1/2 * T0 * G ) - T1 * G = 0
```

T1 * G > 0 and D ^ 2 > 0 so the above can be true only if

```
  ( 13 )    D ^ ( k - 2 ) - 1/2 * T0 * G > 0
```

This means that a smallest k value can always be found to a specific D but k cannot be constant: if D increases then k also has to be increased.
The reason of that is G (D) increases faster than D ^ (k - 2):
- in case of queen: G (D) can be overestimated as D! / (2 ^ D)
- in case of rook: G (D) is exactly D!
- in case of bishop: G (D) is even steeper than D!

2.5.4 So, in cases of k = 1, k = 2 and of general k cases, it can be seen that (7) cannot be true to any D. Thus, 2.1 Assumption is not true.

2.6 We haven't used the not 0 time needed by the generation of Good placings and not Good placings. This is one further element into the (6) expression which depends on the D and let it be

```
  ( 14.a )  TS = TS ( D ) != 0
```

According to (3.b), using large D values

```
  ( 14.b )  TS ( D ) < TS ( D + 1 )
```

and, we can see from 2.5.3 the tendency of TS (D) (which is at least the tendency of the G (D) because Good and not Good placings are always possible and there will be more and more)
So, this further element cannot decrease the tendency of (6), it can increase instead. In this way, the 2.5.4 remains true if TS (D) != 0.

3. CONCLUSIONS

3.1 The above logic tells us that quick and Running time accordingly to 1.10 is not possible while running the N figure problem solver implementation of A algorithm.

3.2 It can be seen from (13) that 2.1 would be true in case of T0 = 0 to any D using a constant k value. But having this, the implementation of A could decide a Good placing within 0 time which supposes infinitely fast computer, but this is impossible to build by our current knowledge.

3.3 We didn't consider the queen kind of pieces to put onto the Chess board so 2.5.4 remains true to use any classical kind of chess figure.