ПЛОВДИВСКИ УНИВЕРСИТЕТ "ПАИСИЙ ХИЛЕНДАРСКИ", БЪЛГАРИЯ
НАУЧНИ ТРУДОВЕ, ТОМ 35, КН. 3, 2007 – МАТЕМАТИКА
PLOVDIV UNIVERSITY "PAISSII HILENDARSKI", BULGARIA
SCIENTIFIC WORKS, VOL. 35, BOOK 3, 2007 – MATHEMATICS

A REFLEXIVE ALGORITHM FOR THE ROOK PROBLEM

Dobromir P. Kralchev, Dimcho S. Dimov, Alexander P. Penev

Abstract. We propose a new, heuristic algorithm for the rook problem. The algorithm is reflexive: it examines its own running-time, which is in correlation with the output.

Key words: rook problem, perfect matchings, binary matrices, heuristics, reflexive (self-monitoring) algorithms

Mathematics Subject Classification 2000: Primary 68T20; Secondary 68R05, 05A05

1. Essence of the rook problem

You have a chessboard N x N, some of its fields are prohibitted and the others are permitted. Can you put N rooks on the permitted fields of the chessboard so that they do not attack one another?

This is usually called the *rook problem*. Many real-world problems can be reduced to it: assigning jobs to workers or classrooms to teachers, etc.

The rook problem is equivalent to the perfect matching problem [1].

The next formulation of the rook problem turns out to be most suitable for our purpose: You have a binary matrix N x N. Any N units in different rows and columns form an *assignment*. Can you find at least one assignment?

So the rook problem is a special case of the assignment problem. There are algorithms for the general case — for example, the Hungarian algorithm [2]; but one can use the special features of the binary case to construct faster algorithms, suitable for a big N.

The rook problem consists of three parts:

a) Does at least one assignment exist?

b) How many assignments exist?

c) Find at least one assignment (if there is any).

The first part is especially important for this reason: many problems are OR-compositions of rook subproblems and can be solved following the next schemes:

— examine the subproblems one by one trying to find an assignment until you find one or there are no subproblems left;

— examine the subproblems one by one: if there is an assignment, then find it and stop searching, else go to the next subproblem.

The second scheme is faster. That is why, it is important to construct a quick algorithm for the first part of the rook problem. Moreover, the third part can be reduced to the first one [3].

Consider also the fact that N is rather big in practical problems: $N \approx 10^3$. This is the upper limit we want to reach.

To sum it up: our purpose is to find an algorithm that solves the existence part of the rook problem; the algorithm must be fast enough for $N \leq 10^3$.

2. Construction of the algorithm

2.1. Analysis of old algorithms

An algorithm for finding perfect matchings is given in [4].

The second part — about the count of the assignments, resp. the count of the perfect matchings — can be reduced to calculating a permanent. There are approximate formulae in [5]. You can find interesting algorithms for this problem in [6] and [7].

Algorithms for the existence of an assignment are given in [8], [9] and [10].

A detailed analysis of the rook problem can be read in [11] and [12].

We shall accelerate the algorithm from [9]. (In fact, [9] does not contain an explicit formulation of the algorithm, but the theorems, managing different cases, are arranged in the same way as the steps of the algorithm. An explicit formulation of the algorithm is given in [11] and [12].)

A *rectangle of zeros* is any submatrix containing only zeros. The basic idea in [9] is that you can search for a big rectangle of zeros instead of an assignment.

Theorem 1. *There is an assignment in a binary matrix of a size N x N if and only if there is not a rectangle of zeros of a size P x Q with $P + Q > N$.*

Searching for a big rectangle of zeros, the algorithm checks each zero for a possible participation in such a rectangle. To do so, the algorithm explores some submatrices (according to the position of the zero being checked) and makes recursive calls when necessary.

The details of the algorithm are unimportant for the current study. You can find them in [9], [11] and [12].

Density (%)	10	20	30	40	50	60	70	80	90
'Yes'	42	46	50	53	56	28	24	18	14
'No'	12	13	14	14	15	20	18	16	13

Average running-time (in centiseconds) for a 1000 x 1000 matrix.

The *density* is the percentage of the units of the matrix. The last two rows contain the average running-time depending on the density of the matrix and the output of the algorithm ('Yes' means there is an assignment, 'No' means there is not).

Obviously, the running-time of the algorithm depends on its output: the positive answer ('Yes') consumes more time. So the algorithm can be accelerated through restricting its running-time. A new algorithm is thus obtained consisting of two levels: the lower level is the old algorithm, the higher level is a monitor that can stop the low-level algorithm, if it consumes too much time. The levels are united by a common goal, so they form a single self-monitoring algorithm, hence the name 'reflexive'.

2.2. A reflexive algorithm for the rook problem

```
Input:
  A: the binary matrix of a size N x N;
  T: the maximal running-time allowed.
Question: Does there exist an assignment in the matrix A?
Output: 'Yes' or 'No' — the answer to the question.
Actions (of the higher level):
1. Run the low-level algorithm that searches A
   for a rectangle of zeros of a size P x Q with P + Q > N.
2. Set a timer to measure out T seconds.
3. Wait for the lower level to finish
   or for the timer to fire.
4. If the lower level finishes within T seconds,
   stop the timer and return the answer of the lower level.
5. If time is up, terminate the lower level
   and return a positive answer ('Yes').
```

The longer T, the less the probability for a wrong answer, but the longer the running-time of the new algorithm. The reasonable values of T are between 0.20 sec. (the maximal value at the 'No' row of the table) and 0.30 sec. (the average running-time for a 1000 x 1000 matrix, regardless of the output). These values of T should decrease the running-time twice.

Of course, the actual running-time depends on the hardware and the size N of the matrix, so the value of T must be different for each N and must be chosen after having the actual running-time of the lower level tested. It is easy to implement the testing procedure in the higher level so that T becomes a variable whose value is dynamically adjusted during a series of calls.

It is possible to make the actions of the higher level dependent on the density of the matrix. The correlation between the running-time and the answer of the lower level is stronger for rare matrices (of a density up to 50%). The higher level could first inspect the density ρ of the matrix A and set the timer only if $\rho \leq 50\%$.

3. Conclusion

Reflexive algorithms are suitable when the output is in correlation with some easily recognizable characteristic of the algorithm. Running-time can often be used in this part, but it is not one and only. Other characteristics, such as memory consumption, may also be useful.

References

[1] Alex Sivkins, Approximate Counting, CS 683: Advanced Algorithms, 2001, `www.cs.cornell.edu/Courses/cs683/2001SP/lec17.ps`

[2] Dimiter Ivanchev, The assignment problem, *"Mathematics" magazine*, No. 9-10, Sofia, 1990, pp. 10–18 (in Bulgarian).

[3] Dobromir Kralchev, Dimcho Dimov, Alexander Penev, Generating an assignment in the rook problem, *"Mathematical forum" magazine*, vol. 5, No. 1, Sofia, 2003, pp. 12–15 (in Bulgarian).

[4] Preslav Nakov, Fundamentals of computer algorithms, TopTeamCo, Sofia, 2001, ISBN: 954-8905-04-3, pp. 175–179 (in Bulgarian).

[5] Dobromir Kralchev, Dimcho Dimov, Alexander Penev, Estimation of the permanent of a binary matrix, *WSEAS International Conference on Applied Informatics and Communications*, Rhodes Island, Greece, November 15–17, 2003 (invited paper).

[6] Avi Widgerson, Computational Complexity Theory, 1999, `http://www.cs.huji.ac.il/~amirs/complex/ex/ex4.ps`

[7] Mark Jerrum, Alistair Sinclair, Eric Vigoda, A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries, *Electronic Colloquium on Computational Complexity*, Report 79 (2000), pp. 1149–1178, `http://citeseer.nj.nec.com/532431.html`

[8] Dobromir Kralchev, Dimcho Dimov, Alexander Penev, An algebraic method for solving the rook problem, *Scientific Works*, vol. 33, book 3, pp. 53–60, 2001 — Mathematics, Plovdiv University "Paissii Hilendarski", Bulgaria.

[9] Dimcho Dimov, Dobromir Kralchev, Alexander Penev, Stanimir Stanchev, Existence of solutions to the assignment problem, *International Conference on Automatics and Informatics*, Sofia, May 30 – June 2, 2001, pp. I-81 – I-83.

[10] Dobromir Kralchev, Dimcho Dimov, Alexander Penev, Statistical analysis of the rook problem, *"Mathematical forum" magazine*, vol. 4, No. 4, Sofia, 2002, pp. 106–110 (in Bulgarian).

[11] Dobromir Kralchev, Investigation of the rook problem, BSc degree paper, Plovdiv University "Paissii Hilendarski", Department of Mathematics and Informatics, Plovdiv, Bulgaria, 2001 (in Bulgarian).

[12] Dobromir Kralchev, Existence, generation and count of the assignments in the rook problem, MSc degree paper, Plovdiv University "Paissii Hilendarski", Dept. of Mathematics and Informatics, Plovdiv, Bulgaria, 2003 (in Bulgarian).

Dobromir P. Kralchev
University of Food Technologies
Dept. of Informatics and Statistics
26 Maritsa Blvd.
4000 Plovdiv, Bulgaria
e-mail: dobromir_kralchev@abv.bg

Received 09 July 2006

Dimcho S. Dimov, Alexander P. Penev
"Paissii Hilendarski" University
Dept. of Mathematics and Informatics
236 Bulgaria Blvd.
4000 Plovdiv, Bulgaria
e-mail: apenev@pu.acad.bg

РЕФЛЕКСИВЕН АЛГОРИТЪМ ЗА ЗАДАЧАТА ЗА ТОПОВЕТЕ

Добромир Кралчев, Димчо Димов, Александър Пенев

Резюме. Предлагаме нов, евристичен алгоритъм за задачата за топовете. Алгоритъмът е рефлексивен: наблюдава своето време за изпълнение, което се намира във взаимна връзка с резултата.