Using grid systems for enumerating combinatorial objects on example of diagonal Latin squares

E. I. Vatutin[1,a], O. S. Zaikin[2], A. D. Zhuravlev[3], M. O. Manzyuk[3], S. E. Kochemazov[2], V. S. Titov[1]

[1] Southwest State University, Kursk, Russia

[2] Matrosov Institute for system dynamics and control theory of SB RAS, Irkutsk, Russia

[3] Internet portal BOINC.ru, Moscow, Russia

E-mail: [a] evatutin@rambler.ru

In this paper we consider the problem of enumerating diagonal Latin squares of small order. In particular we discuss possible algorithmic approaches to this problem and show our results in this regard. Surprisingly, our research showed that the best algorithm for enumerating diagonal Latin squares consists of a number of fixed loops, and its effectiveness can be significantly increased by careful tuning and applying special heuristics. We used the constructed algorithm to enumerate all diagonal Latin squares of order 8. Also, it is being used to carry out large-scale computational experiment aimed at enumeration of diagonal Latin squares of order 9.

Keywords: combinatorics, enumeration, Latin square, volunteer computing

The research was partially supported by Russian Foundation for Basic Research (grants 14-07-00403-a, 15-07-07891-a and 16-07-00155-a) and by Council for Grants of the President of the Russian Federation (grants NSh-8081.2016.9, MK-9445.2016.8 and stipend SP-1184.2015.5). We thank citerra [Russia team] for his help in development and implementing of some algorithms.

Introduction

One of the important classes of combinatorial and discrete optimization problems [Colbourn, Dinitz, 2006] is formed by enumeration problems. During their decision one needs to determine how many objects with specified properties exist. The simplest examples of such problems are the well-known problems about chess rooks, chess queens, etc. For some of them it is possible to formulate precise analytic decisions, while to solve others one needs to perform exhaustive search to enumerate all possible objects with desired properties. For example, for chess rooks problem the number of possible dispositions of N rooks on the board of size $N \times N$ matches the number of permutations and is equal to $N!$. For some problems from the considered class the number of decisions can be expressed using Stirling numbers (of the first and the second kind), Bell numbers [Becker, Riordan, 1948], the number of combinations or partial permutations and so on. At the same time the precise analytical formulas for the number of decisions for chess queens problem or the number of Latin squares of order N are unknown (in the latter case there are known upper and lower bounds).

The number of decisions usually grows rapidly with the increase of the dimension of a problem N, that is why when enumerating corresponding objects using brute force strategy one has to develop highly effective program implementation that takes into account the features of considered problem and provides high rate of generation of enumerated objects. From the point of view of parallel programming the enumeration problems of such type are weakly coupled problems, thus the algorithms for their solving can be implemented in the form of parallel programs that are efficient within the context of parallel computing environments with various architecture that comprise grid systems.

Enumeration of diagonal Latin squares of order less than 9

In the present paper we study diagonal Latin squares (DLS). An arbitrary DLS is a square table of size $N \times N$, where each cell is filled by an element of some alphabet (typically a number from 0 to $N-1$) in such a way that in each row, each column and also in main diagonal and main antidiagonal all elements are distinct. Basically, DLS are a special case of Latin squares (LS) that satisfy additional diagonality constraints. Using simple transformations that do not violate any of the constraints an arbitrary DLS can be transformed to DLS in which the elements of the first row are sorted in ascending order. The corresponding squares form an isomorphism class of size $N!$. The dependence of number of LS on N is well known and presented by A000315 sequence in the Online Encyclopedia of Integer Sequences (OEIS) [A000315], the dependence of the number of LS with fixed first row has the number A000479. For DLS similar sequences are unknown and apparently can be calculated only using exhaustive search.

One can apply different approaches to generate DLS. For example, in [Vatutin, Zhuravlev, …, 2016] this problem was reduced to Boolean satisfiability problem (SAT) and solved using SAT solvers. In [Vatutin, Zhuravlev, …, 2015] to construct DLS there have been tested exhaustive search methods, randomized search methods, ant colony method, and also the original problem regarding the existence of solution was reduced to discrete combinatorial optimization problem. All tested heuristic methods work relatively well if one wants to construct DLS with more or less uniform distribution over the search space formed by all possible DLS of a specific order. However, they are not really suitable for enumerating DLS because these methods are not complete (i.e. they cannot guarantee that we will construct all possible solutions) and in contrast to the exhaustive search they can produce duplicate solutions over time. The average rate of DLS generation in [Zaikin, Kochemazov, 2015] for exhaustive search was about 1 DLS of order 10 per second. In [Zaikin. Vatutin. …, 2015] there was proposed to fill the cells of DLS in a specific order, in particular, to fill diagonals first and then all the other elements row by row. It made it possible to improve the generation rate to about 5000 DLS per second. In [Vatutin, Zhuravlev, …, 2015] we performed several algorithmic and high-level optimizations of the exhaustive search where while filling Latin square cells we prioritize the ones for which on the current step the number of possible elements is low. This and several other modifications made it possible to increase generation rate to approximately 200 000 DLS of order 10 per second in a recurrent implementation. When we reorganized the algorithm in the form of iterative implementation with N^2 nested loops, the generation rate increased to 340 000 DLS per second.

An advantage of recurrent implementation lies in the fact that we can modify the order in which we fill non diagonal cells based on the number of possible elements that can be put into them. However, commands CALL and RET in the program code reduce the performance of the program implementation due to frequent returns. The iterative implementation does not possess this disadvantage, but for a price: the order of loops in it is fixed at compile time and can not be changed. Despite this fact, its performance is about 1.7 times better, thus it looks more promising for future work.

In the present paper we further studied how the order of filling cells influences the generation rate. In particular, we managed to find the order which significantly increases the generation rate compared to the previously used one. To produce it we used a simple heuristic: on each step we choose to fill the cell, which is the most “constrained” by already filled cells. It is based on a simple observation that for an arbitrary cell, the size of set of values that can be put into it decreases with each already filled cell in the same row/column and diagonal (for diagonal cells). Here we do not care about particular values, only about the fact that the cells are filled. Thus, we fix the first row of the DLS (since any DLS can be transformed accordingly) and choose the cell to be filled next in an iterative process. It can be best explained using the following figure, on which we consider DLS of order 5 as an example.

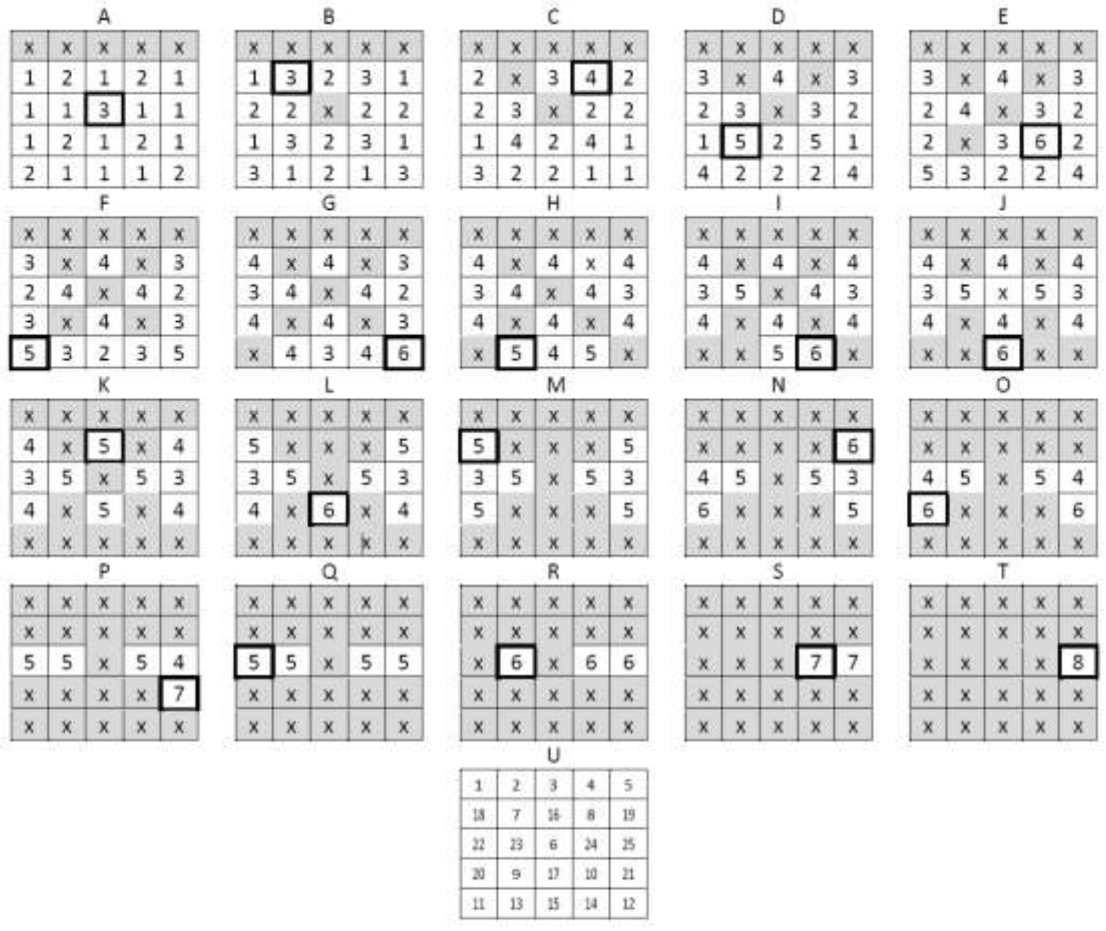

Fig. 1. The order of filling cells for DLS of order 5. Crosses correspond to already filled cells, numbers correspond to the number of constraints imposed by other cells

On Fig. 1A we show the initial condition of Latin square cells. Grey cells marked with 'x' correspond to already filled cells from the first row. It means that every other cell of this square has at least one constraint (because in the same column one element is already used). For cells positioned on main diagonal or main antidiagonal there are two constraints: one imposed by the filled element in the same column and one produced by the element from the diagonal. It means that the cell in the third row and third column has the largest number of constraints (3), thus on this step we fill it. The situation after filling the cell (3,3) is displayed on Fig. 1B. Here we recalculate the number of constraints, choose the most constrained cell, etc. In a situation when several cells have the same amount constraints we choose one of them at random. As a result we construct the order displayed on Fig. 1U. On the first glance it may seem that this order does not significantly differ from the previously used one (where we first fill in both diagonals, and then remaining cells row-by-row), however, the generation speed increased significantly: up to 790 000 DLS per second. This implementation was used to enumerate all DLS for $N < 9$. The corresponding results are shown in Table 1.

Table 1. Number of DLS of order N

N	Number of DLS with fixed first row	$N!$	Total number of DLS	Computing time on 1 CPU core
1	1	1	1	< 1 second
2	0	2	0	< 1 second
3	0	6	0	< 1 second
4	2	24	48	< 1 second
5	8	120	960	< 1 second
6	128	720	92 160	< 1 second
7	171 200	5 040	862 848 000	2 seconds
8	7 447 587 840	40 320	300 286 741 708 800	30 hours

On the enumeration of diagonal Latin squares of order 9

After obtaining results which were described Section 2, we managed to improve the performance of the algorithm by three more heuristics. Let us briefly describe them below.

The first heuristic influences the order of cells. In particular, let us consider the situation depicted on Fig. 1E (see Section 2). It is easy to see that at this state there is only one element on the main antidiagonal that is not filled. Since we consider diagonal Latin squares, it means that there is at most one possible element to be put into the cell (5,1). However, in the previously outlined order, as the next cell to be filled we chose (4,4). We found that if we process such situations by filling the last unfilled cell in row/column/diagonal right when such situation arised, we can improve the algorithm effectiveness.

The second heuristics is tied to the first in that it works with last unfilled elements in row/column/diagonal. From the definition of diagonal Latin square we know, that the sum of all elements within a single row (similar for columns and diagonals) is equal to $(N-1)\times\frac{N}{2}$. It means that we can use formula to compute the value of a single unfilled element without the need for iterating over all possible values.

The third heuristic is a kind of lookahead, commonly used in combinatorial algorithms [Golomb, Baumert, 1965]. Its general idea is to sometimes spend a little more resources than necessary in order to avoid spending much more in future. It depends on several parameters which require specific tuning. Let us consider it in more detail. Remind that we fill the cells of diagonal Latin square one by one in a specific order. It means that on certain stages of this process there arise situations when the number of constraints on some cells is so large, that it can completely eliminate all possible element values, thus making further search useless (e.g. more than 9 constraints when constructing DLS of order 9). However, if we check too often and for too many cells we did not fill yet, the corresponding actions may significantly reduce the performance of the algorithm. We found out that when we apply these restrictions on steps from 42 to 60 for DLS of order 9 we have the most profit.

The described three heuristics make it possible to improve the enumeration speed for DLS of order 9 to about 1 800 000 DLS per second. While this is a significant success, it is clear from the analysis of Table 1, that if we want to compute the number of diagonal Latin squares of order 9, we need to employ high performance computing, because it is unrealistic to do it on a single PC. Thus we started the experiment in volunteer computing project Gerasim@home that is aimed at enumeration of all diagonal Latin squares of order 9.

References

Colbourn C.J., Dinitz J.H. Handbook of Combinatorial Designs // Second Edition. Chapman&Hall. — 2006. — 984 p.

Becker H.W., Riordan J. The arithmetic of Bell and Stirling numbers // American Journal of Mathematics. — 1948. — Vol. 70. — P. 385–394.

Number of reduced Latin squares of order n; also number of labeled loops (quasigroups with an identity element) with a fixed identity element. [Electronic resource]. URL: https://oeis.org/A000315.

Vatutin E.I., Zhuravlev A.D., Zaikin O.S., Titov V.S. Employing algorithmic features of the problem for generation of diagonal Latin squares // Proceedings of the South-West State University. Series 'Control, Computer Engineering, Information Science. Medical Instruments Engineering'. — 2016. — No. 2 (65). — P. 46–59.

Vatutin E.I., Zhuravlev A.D., Zaikin O.S., Titov V.S. Features of the use of weighting heuristics in the search for diagonal Latin squares // Proceedings of the South-West State University. Series 'Control, Computer Engineering, Information Science. Medical Instruments Engineering'. — 2015. — No. 3 (16). — P. 18–30.

Zaikin O.S., Kochemazov S.E. The search for pairs of orthogonal diagonal Latin Squares of order 10 in the volunteer computing project SAT@home // Bulletin of the South Ural State University: Series 'Computational Mathematics and Software Engineering'. — 2015. — Vol. 4, No. 3. — P. 95–108.

Zaikin O.S. Vatutin E.I. Zhuravlev A.D., Manzyuk M.O. Applying high-performance computing to searching for triples of partially orthogonal Latin squares of order 10 // Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies. Arkhangelsk, Russia, March 29-31, 2016. CEUR-WS. — 2016. — Vol. 1576. — P. 155–156.

Golomb S.W., Baumert L.D. Backtrack Programming // Journal of the ACM. — 1965. — Vol. 12, Issue 4. — P. 516–524.