RANDOMIZED PARALLELIZATION – A NEW METHOD FOR SOLVING HARD COMBINATORIAL PROBLEMS

Arkadij Zakrevskij

***Abstract**: A new method for solving some hard combinatorial optimization problems is suggested, admitting a certain reformulation. Considering such a problem, several different similar problems are prepared which have the same set of solutions. They are solved on computer in parallel until one of them will be solved, and that solution is accepted. Notwithstanding the evident overhead, the whole run-time could be significantly reduced due to dispersion of velocities of combinatorial search in regarded cases. The efficiency of this approach is investigated on the concrete problem of finding short solutions of non-deterministic system of linear logical equations.*

***Keywords**: combinatorial problems, combinatorial search, parallel computations, randomization, run-time, acceleration.*

***ACM Classification Keywords**: G.2.1 Combinatorics – combinatorial problems, combinatorial search, G.3 Probability and Statistics – randomization, G.4 Mathematical software – efficiency, parallel and vector implementations.*

Introduction

There exist some hard combinatorial optimization problems that could be solved on computer by many different ways, and it is practically impossible to say in advance which of these ways will lead to the solution quicker. Moreover, the run-times for these ways may differ very much, in tens, hundreds, thousands and more times. Unfortunately, we do not know beforehand, which of them is better for the concrete input data.

In such a case, it would be more profitable to use several competing ways simultaneously and organize parallel computations looking for the solution, terminating them as soon as one of the competitors will find the solution.

In other words, a problem under solution could be changed for a selection of other problems solved in parallel. These new problems could seem quite different, but nevertheless they should be equivalent to the initial problem, that means they will have the same set of solutions. The overhead expenses for constructing these sets of new problems and dealing with them can be compensated with interest by the essential acceleration of the search for solution of the considered problem.

That idea is demonstrated below by the problem of finding a short solution of an undefined system of linear logical equations.

Solving Linear Logical Equations

Let us regard a system of *m linear* Boolean equations with *n variables*

$$\begin{aligned} &a_1^1x_1 \oplus a_1^2x_2 \oplus \ldots \oplus a_1^nx_n = y_1 ,\\ &a_2^1x_1 \oplus a_2^2x_2 \oplus \ldots \oplus a_2^nx_n = y_2 ,\\ &\qquad\ldots\\ &a_m^1x_1 \oplus a_m^2x_2 \oplus \ldots \oplus a_m^nx_n = y_m , \end{aligned} \tag{1}$$

where a_i^j are the Boolean coefficients of the system, x_j – unknown variables (a_i^j, $x_j \in \{0, 1\}$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$), $\oplus$ is the EXOR operator.

This system can be represented in the compact matrix form:

$$A\,x = y\,, \tag{2}$$

where A is a Boolean $m \times n$ coefficient matrix, $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$ – Boolean vectors. AND operator is used as an internal one, and EXOR as the external one for the matrix multiplication.

$$\bigoplus_{j=1}^{n} a_i^j x_j = y_i . \tag{3}$$

Solutions (roots) of this system are the sets of variable values, for which all equations will be satisfied. In this case EXOR sum of the columns of matrix A, corresponding to the component-wise sum by modulo 2 of the columns of A, which correspond to the variables having value 1, should be equal to the column y. The system can be deterministic (defined, having the single solution), non-deterministic (undefined, several solutions exist) or contradictory (over-defined), where there are no solutions [1, 2]. Let us assume below that $n > m$ and all rows of A are linearly independent, so the system is non-deterministic. Then the number of possible solutions is equal to 2^{n-m}.

A great property of the linear equations is that they can be solved in regard of any variable. An effective Gauss method of variable exclusion is based on this property [3] and allows finding the only solution for a deterministic system or the set of all possible solutions for a non-deterministic one. But it is not sufficient when some optimal solution should be selected from that set which could be very large.

For instance, solving a non-deterministic system of linear logical equations $A\,x = y$, it is necessary sometimes to find a solution with minimum number of unknown variables $x_1, x_2, \ldots, x_n$ taking value 1 – such a solution is called *shortest*. That is important in many practical cases, for instance, when logic circuits in AND/EXOR basic are synthesized [4-6] or when some problems of information security are considered.

Looking for Short Solutions of Systems of Linear Logical Equations

Evidently, a shortest solution could be found by means of selecting from matrix A one by one all different combinations of columns, consisting first of 1 column, then of 2 columns, etc. and examining each of them to see if their sum equals vector y. As soon as it happens, the current combination is accepted as the sought-for solution. That moment could be forecasted. If the weight of the shortest solution (the number of 1s in vector x) is w, the number N of checked combinations is defined approximately by the following formula, where C_n^i is the number of i-element subsets taken from a set of n elements:

$$N = \sum_{I=0}^{w} C_n^i . \tag{4}$$

It could be very large, as is demonstrated below.

It was shown [7], that the expected weight γ of the shortest solution of an SLLE with parameters m and n can be estimated before finding the solution itself. We represent this weight as a function $\gamma(m, n)$. First we find the mathematical expectation $\alpha(m, n, k)$ of the number of solutions with weight k. We assume that the considered system was randomly generated, which means that each element of A takes value 1 with the probability 0.5 and any two elements are independent of each other. Then the probability that a randomly selected column subset in matrix A is a solution equals 2^{-m} (the probability that two randomly generated Boolean vectors of size m are equal). Since the number of all such subsets having k elements equals $C_n^k = \dfrac{n!}{(n-k)!\,k!}$, we get:

$$\alpha(m, n, k) = C_n^k\, 2^{-m}. \tag{5}$$

Similarly, we denote as $\beta(m, n, k)$ the expected number of the solutions with weight not greater than k:

$$\beta(m, n, k) = \sum_{i=0}^{k} C_n^i\, 2^{-m}. \tag{6}$$

Now, the expected weight γ of the shortest solution can be estimated well enough by the maximal value of k, for which $\beta(m, n, k) < 1$:

$$\gamma(m, n) = k, \qquad \text{where } \beta(m, n, k) < 1 \le \beta(m, n, k+1). \tag{7}$$

For example, for a system of 40 equations with 70 variables the expected weight γ equals 10, and reaches 31 when m = 100 and n = 130.

In the last case, the described above simple algorithm should check about 10^{30} combinations, according to formula (4) in which $w = \gamma$. Examining them with the speed of one million combinations per second we need about 30 000 000 000 000 000 years to find the solution. Practically impossible!

A great acceleration can be achieved by using Gaussian method of variables exclusion [3] developed for solving systems of linear equations with real variables and adjusted for Boolean variables [5]. It enables to avoid checking all subsets of columns from A which have up to w columns, when only one of 2^{n-m} regarded combinations presents some roots of the system.

Its main idea consists in transforming the extended matrix of the system (matrix A with the added column y) to the *canonical form*. A maximal subset of m linear independent columns (does not matter which one) is selected from A and by means of equivalent matrix transformations (adding one row to another) is transformed to **I**-matrix, with 1s only on the main diagonal. That part is called a basic, the rest $n - m$ columns of the transformed matrix A constitute a *remainder*. The column y is changed by that, too.

According to this method the subsets of the remainder are regarded, i.e. combinations selected from the set which has only $n - m$ columns (not all n columns!). It is easy to show that every of these combinations enables to get a solution of the considered system (any sum of its elements can be supplemented with some columns from matrix **I** to make it equal to y). When we are looking for a shortest solution using this method, we have to consider different subsets of columns from the remainder, find the solution for each such subset and select a subset, which leads to the shortest solution. If it is known that the weight of the shortest solution is not greater than w, then the level of search (the cardinality of inspected subsets) is restricted by w. Note that if $w \geq n - m$, then all 2^{n-m} subsets must be searched through.

Now, for the same example (m = 100 and n = 130), $N \cong 10^9$, which means that the run-time of Gaussian method is about only 17 minutes.

Unfortunately (for practical application), it rises very quickly when increasing parameters m and n. For example, when m = 625 and n = 700, the number of checked combinations reaches 2^{75} and the run-time surpasses one milliard years.

It could be significantly reduced when we know that there exists a short solution with weight w less than γ. For example, the following situation could be imagined. A random matrix A is generated, then w columns are selected by random, and their component-wise modulo 2 sum is defined as vector y.

In that case the search (checking different solutions one by one in order to find a shortest one) could be interrupted as soon as the weight v of the current solution satisfies the inequality $v < \gamma$, and we may conclude that the sought-for solution is found. This idea lies in the base of the recognition method [8, 9].

The Level of Search and its Dispersion

Suppose, a short solution exists presented by w columns of matrix A (their sum equals vector y). Let p of them belong to the remainder. Then this solution will be found when such subsets of columns from the remainder are checked which have exactly p elements. In other words, the solution will be found on the *level of search L* when this one equals p.

That quantity depends, first, on the regarded example (how matrix A was generated) and, second, on the way of getting the remainder - how m linearly independent columns were selected from initial matrix A, splitting that matrix into two parts, basic and remainder.

Chances to discover a solution with weight w on the low search level $L = k$ can be estimated as follows. Let us regard an m-component Boolean vector $\mathbf{a}$ having n-component subvector $\mathbf{a}^*$. There exist C_m^w different values of vector $\mathbf{a}$ with w ones. Suppose all of them are equiprobable. The number of those which have exactly k ones in subvector $\mathbf{a}^*$ (by that $k \leq n$) is estimated by formula $C_n^k C_{m-n}^{w-k}$. So, this number comes to 100 $C_n^k C_{m-n}^{w-k} / C_m^w$ percent of the number of values of vector $\mathbf{a}$ with w ones.

An experiment was conducted, where 10 different Boolean matrices *A* were generated at random with parameters n 500 and $m = 430$. In each of them 30 different subsets of linearly independent columns were selected and corresponding canonical forms were constructed, which produced 300 variants of the remainder. A set of 75 columns was chosen from *A* by random constituting a short solution - their sum was accepted as vector ***y***.

All intersections of that set with remainders were found. The cardinalities of these intersections (numbers of columns in them) define the level of search *L* on which the considered short solution will be found. The values of *L* for all 300 variants are shown in Table 1. Its columns correspond to different examples (*i*) of system and the rows correspond to different remainders in them (*j*) selected at random.

Table 1. Dispersion of level L in space of examples and remainders

j \ *i*	1	2	3	4	5	6	7	8	9	10
1	10	13	12	12	16	11	13	10	12	13
2	8	9	13	9	9	10	11	10	7	9
3	10	10	11	9	11	13	11	10	12	14
4	10	8	11	11	10	8	12	10	13	11
5	10	11	14	6	11	8	17	15	10	6
6	12	6	7	10	12	11	11	12	8	16
7	9	11	8	9	10	12	8	9	8	17
8	8	15	6	13	14	16	15	12	12	12
9	10	13	9	5	14	11	13	6	16	10
10	14	15	9	8	11	10	8	6	13	12
11	14	12	10	12	6	10	12	15	13	9
12	8	9	12	16	16	7	18	11	9	12
13	9	11	6	13	15	9	9	10	10	9
14	14	12	9	14	6	9	7	13	11	13
15	9	5	13	12	11	10	7	13	9	16
16	13	16	6	15	10	7	4	10	11	13
17	6	10	15	13	11	11	13	11	9	12
18	9	9	10	11	10	6	14	8	10	12
19	5	15	7	11	10	16	9	12	9	13
20	11	11	7	8	7	7	11	9	13	10
21	8	12	10	8	8	13	9	6	12	8
22	15	8	11	8	16	6	12	11	15	5
23	10	12	7	14	11	12	13	13	9	9
24	10	9	10	12	4	10	12	11	10	10
25	9	11	10	10	11	11	14	9	7	19
26	11	14	8	10	11	5	14	8	12	10
27	11	12	14	9	10	7	14	11	11	8
28	18	12	13	12	6	8	8	10	12	11
29	12	12	7	9	14	12	7	8	9	12
30	10	7	6	11	13	8	11	15	11	18

As one can see from the table, the level of search *L* is subjected to a great dispersion, varying from 4 up to 19. But this dispersion is surpassed in many times by the dispersion of the corresponding run-times, as is shown below.

The values of run-time for the same examples were obtained, using PC COMPAC Presario, 1000 MH, and presented in Table 2, being measured in seconds (*s*), minutes (*m*), hours (*h*), days (*d*) and years (*y*). The distribution of levels is shown on the right side of the table, *N* indicating the number of entries with value *L* in Table 1.

Note that fulfilled experiments were virtual ones: the run-time was not measured directly but was calculated by the method suggested in [10] and ensuring a rather good approximation.

Table 2. Dependence of run-time *T* on level *L*

L	*T*	*N*	
4	1.6s	2	11
5	22s	5	11111
6	4m	16	1111111111111111
7	38m	16	1111111111111111
8	5h	27	111111111111111111111111111
9	1.5d	36	111111111111111111111111111111111111
10	10d	44	11111111111111111111111111111111111111111111
11	55d	44	11111111111111111111111111111111111111111111
12	280d	39	111111111111111111111111111111111111111
13	3.6y	27	111111111111111111111111111
14	15y	16	1111111111111111
15	59y	12	111111111111
16	214y	10	1111111111
17	713y	2	11
18	2213y	3	111
19	6394y	1	1

Method of Randomized Parallelization

It logically follows from the tables 1 and 2 that efficient algorithms for finding short solutions may be constructed which use the presented great dispersion of values *L* and *T* on the sets of examples and remainders.

These are algorithms of randomized parallelization, which instead of one canonical form of the system (*A*, ***y***) use *q* canonical forms of the same system with different basics selected at random (and remainders defined by them)..Using these forms one by one, such an algorithm consecutively increases the cardinality of checked combinations of columns. That means that it looks for the short solution at first on level $L = 0$, then on level $L = 1$, then on level $L = 2$, etc., until a solution with weight w will be found in some form, which satisfies inequality $w < \gamma$. In that case the algorithm terminates.

Experiments

Some results of the program virtual implementation of that algorithm for different values of *q* are shown in Table 3. Thirty examples of random systems with parameters $n = 1000$, $m = 900$ and the weight of the short solution $w = 100$ have been generated and solved. The following denotation is used in the table:

No is the number of the regarded example,

N is the number of form where the short solution was found,

L is the level at which it was found, and

T is the time spent for it.

Note that value 1 of parameter *q* corresponds to the pure Gaussian method dealing with only one canonical form of the system. Changing it for pseudo-parallel algorithm on the base of 300 forms we accelerate finding the short solution on an average in 46 million times.

The immense acceleration!

Table 3. Results of solving non-deterministic systems of linear logical equations with parameters $n = 1000$, $m = 900$, $w = 100$.

	$q = 1$			$q = 10$			$q = 30$			$q = 300$		
No	*N*	*L*	*T*	*N*	*L*	*T*	*N*	*L*	*T*	*N*	*L*	*T*
1	1	8	7d	9	6	16h	9	6	18h	33	4	19m
2	1	9	69d	8	6	14h	16	5	2h	254	2	4m
3	1	10	2y	7	7	7d	28	6	2d	234	2	4m
4	1	13	776y	3	6	5h	3	6	8h	117	3	5m
5	1	3	4s	1	3	4s	1	3	5s	1	3	4m
6	1	10	2y	5	7	5d	26	5	3h	56	2	4m
7	1	12	112y	9	4	3m	9	4	3m	57	3	5m
8	1	8	7d	10	5	1h	10	5	1h	106	3	5m
9	1	12	112y	10	5	1h	28	4	10m	141	3	6m
10	1	9	69d	2	6	4h	2	6	6h	95	2	4m
11	1	13	776y	7	8	82d	15	4	5m	50	2	4m
12	1	13	776y	9	8	104d	14	5	2h	117	3	5m
13	1	10	2y	8	6	14h	8	6	16h	35	3	4m
14	1	6	58m	1	6	2h	1	6	4h	39	3	4m
15	1	6	58m	1	6	2h	11	5	1h	134	3	6m
16	1	14	4954y	2	8	27d	28	4	10m	205	3	7m
17	1	6	58m	1	6	2h	14	5	2h	49	3	4m
18	1	10	2y	2	7	2d	27	5	3h	285	2	4m
19	1	13	776y	8	7	8d	8	7	9d	84	3	5m
20	1	10	2y	2	6	4h	2	6	6h	203	4	1,3h
21	1	8	7d	7	5	45m	7	5	52m	93	4	39m
22	1	7	13h	10	6	17h	27	4	9m	190	3	6m
23	1	12	112y	2	5	13m	16	2	4s	16	2	4m
24	1	15	29185y	4	7	4d	24	6	2d	226	2	4m
25	1	10	2y	2	6	4h	2	6	6h	46	3	4m
26	1	13	776y	9	7	9d	16	5	2h	112	4	45m
27	1	10	2y	6	8	71d	16	4	6m	281	3	8m
28	1	12	112y	7	8	82d	18	6	1d	87	3	5m
29	1	12	112y	6	4	2m	6	4	2m	254	2	4m
30	1	12	112y	7	8	82d	22	6	2d	31	4	18m
The sum:			38704y			1,3y			20d			5,3h

Conclusion

A new approach to solving hard combinatorial optimization problems is suggested, demonstrated on the problem of finding a short solution of a non-deterministic system of linear logical equations. Its idea is in changing the regarded problem for a set of other similar problems equivalent to the given one and solving them in parallel. The run-time could be considerably reduced by that, possibly in many millions times, as some results of computer experiments show.

Acknowledgements

The work had been performed thanks to the partly support of ISTC (Project B-986) and the Fond of Fundamental Researches of Belarus.

Bibliography

1. Kostrikin A., Manin Y. Linear algebra and geometry. Translated from the second Russian (1986) edition by M. E. Alferieff. Revised reprint of the 1989 English edition. Gordon and Breach, 1997.
2. Lankaster P. Theory of matrices, Academic Press, New York – London, 1969.
3. Gauss C.F.Beitrage zur Theorie der algebraischen Gleichungen, Gött, 1849.
4. Sasao T. Representations of logic functions using EXOR operators. – In “Representations of discrete functions” (ed. by T. Sasao and M. Fujita) – Kluwer Academic Publishers, Boston/London/Dordrecht, 1996, pp. 29-54.
5. Zakrevskij A.D. Looking for shortest solutions of systems of linear logical equations: theory and applications in logic design. – 2.Workshop "Boolesche Probleme", 19./20. September 1996, Freiberg/Sachsen, pp. 63-69.
6. Zakrevskij A.D., Toropov N.R. Polynomial representation of partial Boolean functions and systems. – Minsk, Institute of technical cybernetics, Belarusian NAS, 2001; Moscow: URSS (Second edition), 2003 (In Russian).
7. Zakrevskij A.D., Vasilkova I.V. Fast algorithm to find the shortest solution of the system of linear logical equations. - Problems of logical design. – Minsk: UIIP of NAS of Belarus, 2002, pp. 5-12. (In Russian).
8. Zakrevskij A.D. Randomization of a parallel algorithm for solving undefined systems of linear logical equations. – Proceedings of the International Workshop on Discrete-Event System Design – DESDes’04. – University of Zielona Gora Press, Poland, 2004, pp. 97-102.
9. Zakrevskij A.D. Efficient methods for finding shortest solutions of systems of linear logical equations. – Control Sciences, 2003, No 4, pp. 16-22. (In Russian).
10. Zakrevskij A.D., Vasilkova I.V. Forecasting the run-time of combinatorial algorithms implementation. – Methods of logical design, issue 2. Minsk: UIIP of NAS of Belarus, 2003, pp. 26-32. (In Russian).

Author's Information

Arkadij Zakrevskij – United Institute of Informatics Problems of the NAS of Belarus, Surganov Str. 6, 220012 Minsk, Belarus; e-mail: zakr@newman.bas-net.by

THE BOUNDARY DESCRIPTORS OF THE *n*-DIMENSIONAL UNIT CUBE SUBSET PARTITIONING[1]

Hasmik Sahakyan, Levon Aslanyan

Abstract: The specific class of all monotone Boolean functions with characteristic vectors of partitioning of sets of all true-vertices to be minimal is investigated. These characteristic vectors correspond to the column-sum vectors of special (0,1)-matrices – constructed by the interval bisection method.

Keywords: monotone Boolean functions, (0,1)-matrices.

ACM Classification Keywords: G.2.1 Discrete mathematics: Combinatorics

1. Introduction

The problem of general quantitative description of vertex subsets of n dimensional unit cube E^n, through their partitions, the existence problem and composing algorithms for vertex subsets by the given quantitative characteristics of partitions are considered. Each of these sub-problems has its own theoretical and practical significance. The existence and composing problems are studied intensively [BI, 1988; C, 1986; S, 1986], but the

[1] The research is supported partly by INTAS: 04-77-7173 project, http://www.intas.be