# Optimization of the Modular Educational Program Structure

Alexandr Ivanchenko, Anastasia Kolomiets, Dmitriy Grinchenkov, Nguyen Van Ngon
Platov South-Russian State Polytechnic University (Novocherkassk Polytechnic Institute)
Information Technologies and Control Department
Prosvescheniya Str. 132, Rostov region, 346428, Novocherkassk, Russia
E-mail: grindv@yandex.ru

***Abstract*—Optimized structure of the educational program consisting of a set of the interconnected educational objects is offered by means of problem solution of optimum partition of the acyclic weighed graph. The condition of acyclicity preservation for subgraphs is formulated and the quantitative assessment of decision options is executed. The original algorithm of search of quasioptimum partition using the genetic algorithm scheme with coding chromosomes by permutation is offered. Object-oriented realization of algorithm in language C ++ is described and results of numerical experiments are presented.**

***Keywords:*** **e-learning, educational program, educational object, computer testing, acyclic weighed graph, number partition, graph partition, random permutation, genetic algorithm, object-oriented approach.**

## I. Introduction

While study processes planning of large volumes of a training material with use of electronic training technologies (for example, for preparation and professional development of the industrial enterprises personnel) all training material of the educational program (EP) is presented usually in the separate portions – educational objects (EO), on each studying completion intermediate certification is carried out (usually in the form of computer testing – CT).

EO are connected by the relation of precedence-consequence, and successful certification for one EO is an admission condition to studying of another EO. Natural model of such EP is the directed acyclic weighted graph $G$=($V$,$E$,$W_E$), where $V$ – a set of n vertex presenting EO, $E$ – a set of edges (precedence- consequence conditions) and – $W_E$ a set of edges weight [1][2][3]. The weight of $w_{ij}$ defines "activation threshold" of an edge: if the assessment on EO $v_i$ is more or equal to $w_{ij}$, studying of EO $v_j$ is possible, otherwise repeated testing on $v_i$ is necessary.

In practice the quantity of EO as a part of EP can be rather big, especially in cases of the corporate educational structures connected with training of employees of the enterprises on various specialties. Under these conditions problem of integration of EO in the educational modules (EM) by graph partition G on a small amount of $k$ subgraphs is becoming actual:

$$V = \cup_{p=1}^{k} V_p, \; \cap_{p=1}^{k} V_p = \emptyset. \tag{1}$$

Rather obvious optimality criteria of such partition are minimization of "interconnections" between EM while preservation of acyclicity for subgraphs.

Before formalization of optimality criteria and condition of acyclicity we will make necessary explanations. We will call the *scheme of graph partition* $s_i(n,k)$ any number partition $n$ into $k$ parts:

$$s_i(n,k) : n = a_1 + a_2 + \cdots a_k. \tag{2}$$

It is known [4, p.71], that total of such partition P is defined recursively:

$$P(n,k) = P(n-1,k-1) + P(n-k,k), \tag{3}$$

where: $P(i,i) = 1, P(i,1) = 1, \forall i, \; P(i,j) = 0, j > i.$

Generation algorithms of all number partition n into k parts are given in [4, p 70]. Thus, when graph partition from $n$ vertex on $k$ subgraphs there is P ($n$, $k$) schemes of partition (for example, $P$ (20,3) = 33).

We will define, how many various options of graph partition for the scheme set (assign $Q(s_i(n,k)$ to that quantity). As there generally can be repeating elements in the scheme of partition, the general view of the scheme will be:

$$s_i(n,k) : n = q_1 a_1 + q_2 a_2 + \cdots q_m a_m, \tag{4}$$

where $q_j$>=1, $m$<=$k$, all $a_j$ varies.

The formula is fair:

$$Q\big(s_i(n,k)\big) = \frac{n!}{q_1!(a_1!)^{q_1} q_2!(a_2!)^{q_2} \ldots q_m!(a_m!)^{q_m}} \tag{5}$$

For example, $s_i$(4,2): 4 = 2+2. Then:

$$Q(s_i(4{,}2)) = \frac{4!}{2!\,(2!)^2} = \frac{2\cdot 3\cdot 4}{2\cdot 4} = 3.$$

It will be such partitions: {(1,2), (3,4)}, {(1,3), (2,4)}, {(1,4), (2,3)}.

One more example: P(8,3)=5. It will be scheme:

$s_1$(8,3)=1+1+6, $s_2$(8,3)=1+2+5, $s_3$(8,3)=1+3+4, $s_4$(8,3)=2+2+4, $s_5$(8,3)=2+3+3

Total number of graph partition from 8 vertex on 3 subgraphs:

$$N(8{,}3) = \sum_{i=1}^{P(8,3)} Q(s_i(8{,}3)) = \frac{8!}{2!6!} + \frac{8!}{2!5!} + \frac{8!}{3!4!} + \frac{8!}{2!(2!)^2 4!} + \frac{8!}{2!2!(3!)^2} = 966.$$

The formula for total number of graph partition from *n* vertex on *k* subgraphs is fair:

$$N(n,k) = \sum_{i=1}^{P(n,k)} Q(s_i(n,k)). \qquad (6)$$

Let us denote $r_l(n,k) = \{V_1^l, V_2^l, \dots V_k^l\}$ for current graph partition. Then we will consider the best for the set *n* and *k* - partition minimizing the following criterion (total weight of interconnections):

$$F(n,k) = \min_{r_l(n,k),\, l\in[1,N(n,k)]} \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}, \quad v_i \in V_x^l,\ v_j \in V_x^l, x \neq y. \qquad (7)$$

It is required to keep an acyclicity condition for subgraphs by its partition. We will consider that partition $r_l(n,k)$ keeps property of acyclicity if for any couple of subgraphs $V_x^l, V_y^l$ all edges connecting them go into one direction.

We will notice that formulated problem of graph optimum partition differs from known graph partition problems [6,7] by the condition that initial graph is directed acyclic and it is required to keep an acyclicity condition for subgraph.

## II. Algorithm for Solution

**Algorithm 1**. Search of optimum graph partition from n vertex on k subgraphs by straightforward enumeration.

[It is supposed that procedure of generation of partition schemes is available (described in literature) and generation procedure of all graph partition for the partition scheme set is available (not presented in literature)]

**Step 1**. Regular scheme of partition is generated.

**Step 2**. The next partition for the current scheme is generated.

**Step 3**. If partition is accepted (provides acyclicity for subgraphs), we pass to a Step 4, else to Step 5.

**Step 4**. The total weight of interconnections for the current partition is calculated and correct the minimum value of criterion.

**Step 5**. If not all partitions for the current scheme are received, transition to a Step 2, else to a Step 6.

**Step 6**. If not all schemes of partition are received, transition to a Step 1, else Stop.

The main shortcomings of this algorithm are: need of development of generation procedure for all graph partitions for the partition scheme set and the exponential growth of algorithm operating time at increase in number of graph vertexes.

It is possible to offer simple algorithm of quasioptimum partition search **(Algorithm 2)**, free from the first shortcoming which is overcome by generation of random permutations of a set of graph vertexes and "appoint" it - the current partition. Of course, there is a problem of repeatability of partition appears, as it is obvious that two various permutations can correspond to the same partition for the scheme set.

For example, for the scheme of 4=2+2 permutation (1,3,2,4) and (4,2,3,1) will be identical because they set the same graph partition into subgraphs with vertexes (1,3) and (2,4). However repeatability of partition can be referred to the property of a quasioptimality of this algorithm. It is obvious that the end condition of the offered algorithm of random permutations (step 5) is performance of the set number of iterations (for example, 10000).

Offered algorithm of random permutations is simplest and it is possible to achieve much better effect from use of random factor in case of application the standard scheme of the genetic algorithm (GA) [7]-[10] which can be "built in" algorithm 1 instead of steps 2-5 (we will call it **Algorithm 3**).

Obvious way of decisions coding (chromosomes on terminology of GA) in the considered task are permutation, as in this case the phenotype and a genotype coincide and coding/decoding operations aren't necessary, and the total weight of interconnections for the current partition will be used as fitness function.

It should be noted that at generation of the next decision (chromosome in the form of permutation) when forming initial population or as a result of performance of mutation/crossing operations it is necessary to carry out two checks:

1) whether this decision is admissible from the point of view of ensuring acyclicity for subgraphs;

2) whether this decision (chromosome) coincides with one of already available in population, or with a chromosome before a mutation, or from one of parental chromosomes.

## III. Practical Realisation of Algorithm

For formalization we will use the formal-language notation on the basis of language C++ that allows to apply exclusively effective in this case object-oriented approach [11-13]. The object model of the considered task is based on several interconnected classes.

Class **Graph** encapsulates a set of *vertex (set<int> vertex)* and weighted edges (*vector<Edge> edges*, where *Edge = pair<pair<int,int>, int>*) and contain in open part necessary functionality:

– function *void push_back(Edge),* including edge in graph,
– function *size_t size(),*returning quantity of vertex of the graph,

- function *bool isEdge(pair<int,int>)*, checking whether there couple of numbers that corresponds to any edge of the graph;
- function *bool one_direction(vector<int>, vector<int>)*, checking whether all edges connecting vertexes of two subgraphs have one direction;
- function *double bind_size(vector<int>, vector<int>),* returning the total power of interconnections of two subgraphs.

Class **Partition** encapsulates partition – array of subgraphs *(vector<vector<int>> P)*. Besides, static fields are stored in open part of this class: the scheme of graph partition (*static vector<int> Scheme*) and the index on graph (*static Graph* G*) that establishes between the classes Partition and Graph the dependence relation. Functionality of the class Partition is provided with the following functions:

- two constructors: by default *Partition()* and constructor of type transformation *Partition(vector<int>),* allowing to initialize a class sample by permutation;
- overloaded operation of assignment *void operator= (vector <int>*), interfaced to the constructor of type transformation;
- overloaded operation of checking of two partitions into identity *bool operator==(const Partition&);*
- function *bool isAcyclic*(), checking partition into acyclicity;
- function *int bind_size(),*returning total power of interconnections for all subgraphs.

For organization of calculation with usage of classes Graph и Partition it is necessary:

- to declare static fields in global area of visibility *vector<int> Partition::Scheme* and *Graph* Partition::G*;
- to create graph in the body of main program *Graph Gr*, fill it with data (edges) through of *Gr.push_back(...)* and to pass a pointer to *Gr* in class Partition: *Partition::G = &Gr*.

In the subsequent it will be necessary to transfer to the class Partition the current scheme of partition by value assignment to the static field *Partition::Scheme*, for example: *Partition::Scheme = {4,5,6}*.

The object model of genetic algorithm includes the following classes:

Class **Chromosome** encapsulates a chromosome as an array of genes *(vector <int> genes)* and an assessment of "fitness" of a chromosome (*double fitness*), as well as the static index on fitness function which will be set in the main program (*static F fit_fun* where *F = double (*) (vector <int> &)*). Functionality of the class Chromosome is provided with the following functions:

- two constructors: by default *Chromosome()* and constructor of type conversion *Chromosome (vector<int>)*, allowing to initialize chromosome by permutation and on the fly and to assess its "fitness" with the help of fitness-function *fit_fun*;
- two functions for “retrieval” values of fields: *vector<int> get_genes()* and *double get_fitness()*;
- overloaded external functions of comparison of two chromosome (==, < and >).

Class **Population** encapsulates population as an array of (*vector<Chromosome> pop*). Functionality of the class Population is provided with the following functions:

- constructors by default *Population()*;
- function of chromosome inclusion in population *void insert(const Chromosome& c)*, which automatically maintains orderliness of chromosomes in population on increase of value of fitness;
- overloaded operation of access to population elements on index *Chromosome operator[](int i);*
- function of obtaining size of population *int size();*
- function of the table frequencies (roulette wheel) creation for population elements *vector<double> get_freq();*
- function of the next population creation *void next_pop().*

The most difficult is the function *next_pop()* which realizes the GA scheme, consistently by following steps:

- select "elite" (the most adapted chromosomes) and includes it in the next population (next generation);
- the rest of the next generation is filled with chromosomes descendants which are formed as a result of application crossing operation (crossover) to two "parental" chromosomes selected from the current population by means of "roulette wheel";
- mutation operation is applied to some chromosomes from the created new generation.

Developed and realized set of classes has allowed to make record of the main algorithm of the problem solution exclusively compact: it is only necessary to create initial population by means of the generator of casual permutation and then the set number of times (for example, 10000) to call the *next_pop* function ().

Fig.1 shows graph from 20 vertexes and its optimum partition into subgraphs A, B and C according to the scheme {5,6,9}. The size of population has been set 50, and the number of iterations (generations) – 500. We will notice that there are 77 597 520 options of partition for that graph according to the scheme {5,6,9}.

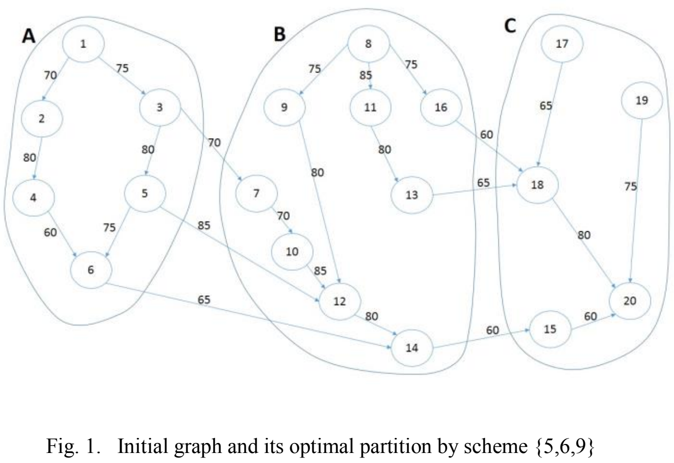

Fig. 1. Initial graph and its optimal partition by scheme {5,6,9}

The integrated representation of initial EP in the form of three modules is shown on the Fig.2. Similar structurization allows optimizing process of training by allocation of a small amount of logically complete educational modules and orienting student to the ordered passing of all set of educational objects [15].

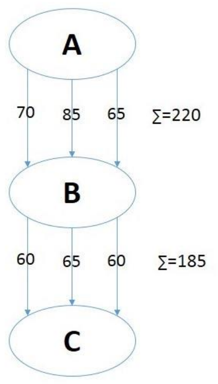

Fig. 2. Graph representation in the form of three subgraphs (modules).

## IV. CONCLUSIONS

The numerical experiments executed by authors on model examples have shown high efficiency of the offered algorithm of creation of quasioptimum graph partition with use of genetic algorithm and coding of decisions by permutations at rather large number of vertexes. Optimized structure of the educational program consisting of a set of the interconnected educational objects is offered by means of problem solution of optimum partition of the acyclic weighed graph. The condition of acyclicity preservation for subgraphs is formulated and the quantitative assessment of decision options is executed. The original algorithm of search of quasioptimum partition using the genetic algorithm scheme with coding chromosomes by permutation is developed.

## REFERENCES

[1] Thanassis Hadzilacos, Dimitris Kalles, Dionysis Karaiskakis, Maria Pouliopoulou. Using Graphs in Developing Educational Material // Proceedings of the 2nd International Workshop on Building Technology Enhanced Learning Solutions for Communities of Practice. TEL-CoPs'07. Sissi, Lassithi Crete. Greece, 18 September, 2007. URL: http://ceur-ws.org/Vol-308/paper04.pdf

[2] Ivanchenko A. N., Nguyen Van Ngon. Simulation modeling of the process study of the modular curriculum // *University News. North-Caucasian Region. Technical Sciences Seri*es. 2015; 3: 28-33

[3] Ivanchenko A. N., Nguyen Van Ngon. Modelling of e-learning using temporal and random graphs // *University News. North-Caucasian Region. Technical Sciences Seri*es. 2016; 1: 15-19

[4] D.L. Kreher and D.R. Stinson, Combinatorial Algorithms: Generation, Enumeration and Search, CRC press LTC , Boca Raton, Florida, 1998. 340p.

[5] W. Kocay and D.L. Kreher, Graphs, Algorithms, and Optimization, Chapman \& Hall/CRC Press, Boca Raton, Florida, 2005.

[6] CME342/AA220/CS238 - Parallel Methods in Numerical Analysis. Graph Partitioning Algorithms. Stanford University. Stanford, California. URL: http://web.stanford.edu/class/cs238/lect_notes/lecture12-13-05.pdf

[7] Gladkov L.A. The bioinspired methods in optimization / L.A. Gladkov, V. V. Kureychik, V. M. Kureychik, P. V. Sorokoletov. – M.: FIZMATLIT, 2009. - 384 pages.

[8] Kureychik, V. V. Theory of evolutionary calculations / V. V. Kureychik, V. M. Kureychik, S. I. Rodzin. – M.: Fizmatlit, 2012. – 260 pages .

[9] Rutkovskaya D. Neural networks, genetic algorithms and indistinct systems / D. Rutkovskaya, M. Pilinsky, L. Rutkovsky. – M.: The hot line – Telecom, 2013. – 384 pages.

[10] Karpenko A.P. Modern algorithms of search optimization. The algorithms inspired by the nature: manual / A.P. The Karpenka – M.: MGTU publishing house of N.E Bauman, 2014. – 446 pages.

[11] Nicolai M. Josuttis. The C++ Standard library: A Tutorial and reference (2nd edition) / Nicolai M. Josuttis // Publisher Addison-Wesley Professional. - 2012. - 1128p.

[12] Musser D.A., Stepanov A.A. Generic Programming / David A. Musser, Alexander A. Stepanov // Proceeding of International Symposium on Symbolic and Algebraic Computation, vol. 358 of Lecture Notes in Computer Science, pp. 13–25, Rome, Italy, 1988.

[13] Stepanov A.A., Mac-Jones P. Beginnings of programming: Transl. from English / A.A. Stepanov, P. Mac-Jones//M.: LLC I.D. Williams, 2011. – 272 pages.

[14] Petrochenkov A. Practical Aspects of Genetic Algorithms' Implementation in Life Cycle Management of Electrotechnical Equipment. Proc. of the 3rd International Conference on Applied Innovations in IT, (ICAIIT), March 2015. URL: http://icaiit.org/proceedings/ 3rd_ICAIIT/1.pdf

[15] Grinchenkov D.V., Kushchy D. N. Methodological, technological and legal aspects of use of electronic educational resources. // *University News. North-Caucasian Region. Technical Sciences Seri*es. 2013. No. 2 (171). Page 118-123.