IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 8, 2022, accepted March 21, 2022, date of publication March 25, 2022, date of current version April 5, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3162215

Optimization of the Functional Decomposition of Parallel and Distributed Computations in Graph Coloring With the Use of High-Performance Computing

JARMILA SKRINAROVA AND ADAM DUDÁŠ
Department of Computer Science, Faculty of Natural Sciences, Matej Bel University, 974 01 Banská Bystrica, Slovakia

Corresponding author: Adam Dudáš (adam.dudas@umb.sk)

This work was supported in part by the High-Performance Computing (HPC) Center of Matej Bel University in Banská Bystrica using the HPC Infrastructure through by the Research & Development Operational Program funded by the Europian Regional Developement Fund (ERDF) (Slovak Infrastructure for HPC) under Project ITMS 26230120002 and Project 26210120002; and in part by The Ministry of Education, Science, Research and Sport of Slovak Republic—Implementation of New Trends in Computer Science to Teaching of Algorithmic Thinking and Programming in Informatics for Secondary Education, under Project KEGA 018UMB-4/2020.

ABSTRACT This article presents methods for correct decomposition for high performance computations related to large sets of graphs. These computations contain large number of calls of sequential, recursive algorithm for NP-complete problem - proper edge coloring of graph. Decomposition of this computational problem is not trivial, since the number of recursions in various parts of the computation is different and causes a high load and time imbalance. We designed, implemented and experimentally verified a new decomposition method that significantly reduces the computational time for a large set of graphs (up to 404 million graphs). This method ensures the same duration of computational time for partial subtasks and thus eliminates the need to wait for synchronization of parallel computations.

INDEX TERMS Decomposition, distributed computing, graph coloring, parallel computing, snark.

I. INTRODUCTION

When designing and implementing massive computing applications for HPC systems, it is necessary to take into account the appropriate allocation of tasks on computing resources. At the same time, it is necessary to decompose the computation so that the individual subtasks of the computation are processed in the similar (in an ideal case equal) time. In this contribution, we compute the proper edge coloring of a large set of graphs (up to 404 million graphs) on an HPC system.

Proper edge coloring of cubic graphs is a problem that is frequently computed as part of gaining knowledge concerning graph data and properties of graphs, in scheduling, in register allocation done as a part of source code translation and compilation, as well as in pattern matching problems [1], [2]. In these problems, the algorithm needs to color many graphs, which represents an NP-complete problem. In this work we will work with cubic graphs - graphs with all vertices of degree 3 - which represent the simplest (non-trivial) instance of the problem of edge coloring of graphs. Although we are working with the simplest instance of the edge coloring problem, in [16] the authors proved that it is still an NP-complete problem.

The novel method presented in this article ensures the optimal decomposition of the problem in the design and implementation of parallel computations to minimize the computational time of edge coloring of a sizable set of graphs. This method consists of meeting two objectives which ensure considerable parallel speedup of computation of given problem - 3-edge coloring of set of graphs.

The first objective is to divide the set of graphs into subsets (clusters) so that for each subset of graphs, it is possible to find the order of edge coloring, which reduces the required time to compute each graph in the subset. The second objective is to find such an order of edge coloring of the graph that the computational time of edge coloring for all graphs in the set is identical (or within a specified tolerance).

The proposed method was verified on large data sets that were used in extensive experiments.

Our motivation for design and implementation of these parallel methods for computing of sequential algorithms for each graph in sizable graph set is to create a suitable size of

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

computational grain [17] in order to use designed algorithms on any computational system [18], [20]. This brings speedup of overall computational time of the problem.

In section II of this paper, we present the research, publications and algorithms related to the problem of reducing the time of cubic graph coloring based on parallel computing and clustering.

Section III contains information regarding load balancing in the system and decomposition of complex tasks into subtasks which acts as main motivation of out work.

Section IV analyzes the theoretical basis for this paper. Concepts of regular edge coloring of graphs and edge 3-uncolorable cubic graphs are presented.

In section V, the edge backtracking algorithm for edge coloring of graphs is described. This algorithm is used as part of our solution for edge coloring of large sets of graphs.

Sections VI - IX contain the main objectives, original methods and algorithms for edge coloring of large sets of graphs, testing of these algorithms and evaluating algorithms while paying attention to correct decomposition of the task.

Conclusions of the presented paper and possibilities for future work related to time reduction of cubic graph edge coloring based on parallel and distributed computing are described in section X.

II. RELATED WORKS

The use of graph coloring as the solution to the scheduling problem is presented in [1]. In the work in [2], the author shows methods of register allocation and spilling by coloring graphs.

Graph coloring is an NP-complete problem that can be solved using several algorithms such as the following:

- Edge-color algorithm presented by the author of [3]. This algorithm uses polynomial space, which improves the previous $O(2^{n/2})$ algorithm of Beigel and Eppstein [4]. [3] uses the natural approach of generating inclusion-maximal matchings of the graph.
- Different approaches to graph coloring were introduced by the authors of [5], who presented a simple but empirically efficient heuristic algorithm for the edge coloring of graphs. The basic idea of this algorithm is the displacement of so-called conflicts (adjacent edges identically colored, see. Fig 3, left) along paths of adjacent vertices whose incident edges are recolored by swapping alternating colors using Kempe interchange.
- A simple backtracking approach to edge coloring of graphs, which uses recursive functions, was presented in several research articles and publications [6]. We describe this algorithm in detail in section V of this paper.

The methods presented in the [3], [5] and [6] are used for sequential edge coloring of singular graphs or sets of graphs. In our work, we aim to edge color large sets of graphs with the use of parallel computation of sequential algorithms for each graph in the sizable graph set in order to minimize the computational time needed for the coloring of graphs. The research in this article is a continuation of research related to the use of parallel and distributed computing to color cubic graphs:

- In paper [7], we briefly introduced the proper edge coloring of graphs in the context of parallel and distributed computations while using various initial edges to color the graph implemented via permutation of the adjacency matrix of the graph.
- The paper [8] presented the use of adjacency matrix permutation to minimize the computation time of proper edge coloring of large sets of graphs. In this work, we present a methodology that can be used to find the order of edges, which reduces the time of computation of edge coloring for certain subset of graphs.
- In [9], we analyze 3-edge coloring of cubic graphs using various initial edges of coloring. We present some of the problematic subgraphs and patterns that increase computing time of edge coloring of graphs. We have also shown the use of adjacency matrix permutation as a mean to work with these patterns and subgraphs more efficiently.

Novel approach presented in this article is based directly on the work and concepts presented in [7]–[9]. These basic concepts are presented in the section V. Novel approaches are presented in the sections $VI - IX$ of this article.

The authors of [10] designed an algorithm for automatic computation and a data decomposition algorithm of a prioritized dominant array (compilation technique that maps computation and data onto different processors), which ranks arrays according to their potential communication costs and finds data decomposition for arrays in decreasing order of rank.

In [11], the authors proposed an improved static decomposition algorithm of distributed memory parallel computers.

Authors of [12] propose an efficient algorithm to solve the constrained data-driven optimization problem with different level of noises. This method is of high efficiency in the comparison with state-of-the-art baseline approaches with multiple noise levels. In [13] authors propose a new dynamical approach to detect the cluster configuration fast and accurately which can be applied to electronic commerce systems. This approach can be viewed as alternative to other - conventional - methods of clustering which do not reach high efficiency of this method.

III. BASIC CRITERIA FOR TASK DECOMPOSITION IN HPC SYSTEMS

High Performance Computing refers to systems facilitating the scaling of applications to a large number of nodes. The characteristic properties of these applications are need for processing of big data or programs containing large number of subtasks.

Therefore, we need to look at the computational application from a global perspective, which on the one hand includes knowledge about the appropriate allocation of tasks on machines of the computing system and on the other hand

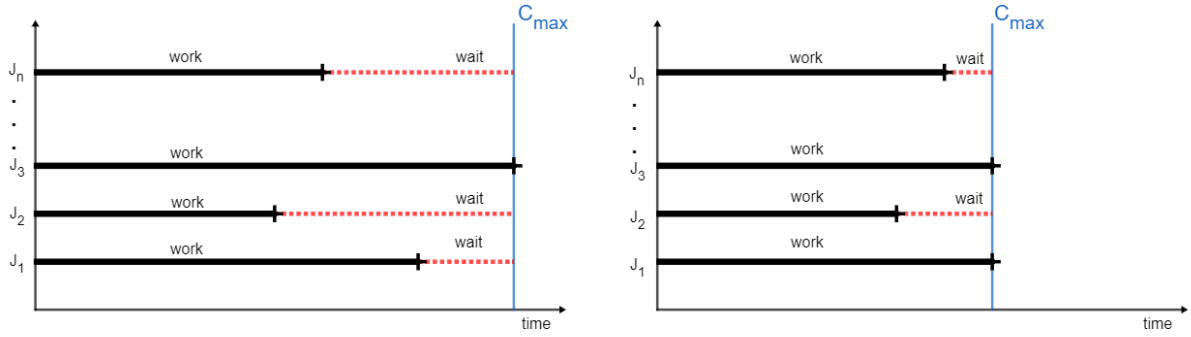

FIGURE 1. Set of jobs with imbalanced load (left) and set of jobs with balanced load (right).

respects the internal structures of the algorithm and the possibilities of its decomposition.

A. LOAD BALANCING IN HPC SYSTEMS

Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of jobs, where job J_i can be described as part of the overall work that cannot be divided further.

Let $M = \{M_1, M_2, \ldots, M_m\}$ be a set of machines, where the machine or computing unit is a set of cumulative resources (processors, memories, and storage space) with limited capacity. [15]

Load balancing is a technique that distributes the load (jobs) between number of machines [18]. Job J_{ij} is the $j-th$ job which needs time for completion of its run on the $i-th$ machine - this completion time of J_{ij} is denoted by C_{ij}. In the case, that completion time of the last job on machine i is C_{max_i}, where $C_{max_i} = max\ \{C_{i1}, \ldots, C_{in}\}$, then completion time of the last job on all machines is denoted by C_{max}, where $C_{max} = max\ \{C_{max_1}, \ldots, C_{max_m}\}$.

A decrease in the computational time of the set of jobs J, which means a decrease of C_{max}, and therefore optimization of computation of the set of jobs J can be reached with the use of load balancing in the computational system.

In the system with imbalanced load the C_{max} grows as a product of large waiting times, therefore time of completion of whole set of jobs J is prolonged (see Fig. 1, left). On the other hand - when the load in the system is balanced, the overall computational time of the set J is decreased (Fig.1, right).

B. DECOMPOSITION OF TASKS IN PARALLEL COMPUTING

When creating a parallel computation, we decompose the complex task P into a set of subtasks $P = \{P_1, P_2, \ldots, P_n\}$. An important parameter that needs to be monitored is the computation time of individual subtasks. In the area of parallel computing there are various types of subtasks - subtasks that can be independent of each other, but more often subtasks which contain various dependencies. A typical dependency between subtasks is created by inserting a synchronization point at which the computation needs to wait for all subtasks to complete. For example, waiting for intermediate results before making a summary computation. [15]

Let $T(Pi)$ be runtime needed to perform the task P_i, then the parallel computation time for the system containing number of processors equal to number of tasks is given by the maximum of time intervals $T(P_i)$ [18]:

$$T(n, P) = t_c - t_s = max\{T(P_i)|i = 1 \ldots n\}, \quad if\ p = n \qquad (1)$$

where t_c denotes time of completion of the subtask and t_s denotes time of start of computation related to the subtask. Therefore, for a correct decomposition, we require approximately identical time intervals *T(Pi)* to solve individual subtasks [18], [19]:

$$T(P_1) \approx T(P_2) \approx \ldots \approx T(P_n) \qquad (2)$$

Since we often do not have a system that is wide enough - it does not contain as many processors as necessary - we need to process tasks in batches (see Section VIII).

No matter what variant of parallelism is used, it may not be known at compile time how much time a part of work actually takes. In general, an algorithm that requires each worker to perform a number of iterations or recursions in order to reach some convergence limit could be inherently time imbalanced, since a different number of iterations may be needed for each part of the computed problem. [14] Therefore, it is not trivial to achieve identical computation times specified in relation (2).

As conclusion of this section of the paper, we summarize the basic criteria for task decomposition in HPC systems:

- load balancing in High Performance Computing - decrease of C_{max} in batches of jobs,
- decomposition of task in such a way, that criteria for time balancing are fulfilled.

This paper aims to present parallel computations for edge coloring of sets of graphs, the basis of which is the correct decomposition of a given problem. As can be seen in the section V, these jobs may contain recursions (with an unpredictable number of recursive iterations) and create an imbalance of computational time.

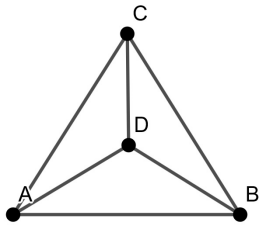

FIGURE 2. Example of a simple graph.

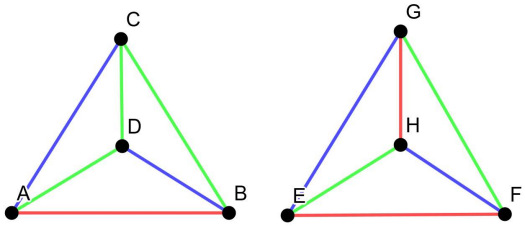

FIGURE 3. Improper (left) and proper (right) coloring of the same graph.

IV. EDGE COLORING OF SNARKS

The problem of proper edge coloring of cubic graphs, described in this section of the paper, is classified as an NP-complete problem [16]. Graph G consists of the following [21]:

- vertices: elements of set $V(G)$. In Fig. 2, vertices are labeled with capital letters A, B, C and D.
- edges– elements of set $E(G)$. An edge is a connection between two vertices, so we can label it using the names of these two vertices.

Graph G is a pair of sets V and E, where elements of set E are double-element subsets of set V [22]:

$$G = (V, E), \quad E \subseteq [V]^2 \tag{3}$$

If vertex V is of degree 3, we use $deg(V) = 3$. This concept represents the number of edges that are incident to a given vertex (since we strictly work with undirected graphs, by incident, we mean connected to the vertex in any way). The highest (maximal) degree of vertex in graph G is denoted by $\Delta(G)$. In this paper, we consider strictly cubic graphs. The graph is cubic when all of its vertices are of degree 3 (see Fig. 2).

A. EDGE COLORING OF GRAPHS

The edge coloring of the graph is an operation of assigning colors to individual edges of the graph. Coloring is called *proper* when there is no conflict in the coloring of a given graph, which implies that no vertex of a given graph is incident to two or more edges with the same color. The lowest number of colors usable in the proper edge coloring of graph G is called the edge chromatic index of graph G, and it is denoted by $\chi'(G)$ [22].

Vizing's theorem [22], which says that the minimal number of colors required to color the graph is in the interval $\langle \Delta(G), \Delta(G) + 1 \rangle$, holds true. The formal notation of Vizing's theorem focuses on the minimal number of colors:

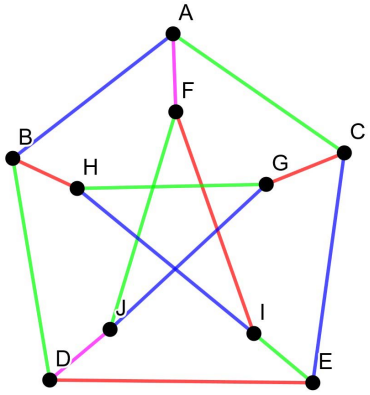

FIGURE 4. Proper coloring of smallest snark - Petersen graph.

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1 \tag{4}$$

where $\Delta(G)$ is the maximal degree of the vertex in graph G, and $\chi'(G)$ is the edge chromatic index of graph G. Since every vertex of the cubic graph is of degree 3, we consider three or four colors to properly color the cubic graph.

B. EDGE 3-UNCOLORABLE CUBIC GRAPHS

If anybody chooses random graph G from the group of cubic graphs, the chromatic index of this graph is likely equal to 3. This cubic graph G is called edge 3-colorable [23]. There is a small group of cubic graphs that need four colors for their proper coloring. Graphs from this group of cubic graphs are called edge 3-uncolorable or snarks [6].

The chromatic index of snarks is $\chi'(G) = 4$. To determine whether a given graph G is a snark, we must edge color it using three colors. Therefore, the coloring algorithm needs to check every possibility of edge 3-coloring of graph G. Algorithms can decide whether the graph is snark after checking all possible edge colors using three colors.

V. EDGE BACKTRACKING ALGORITHM

This section introduces the sequential algorithm based on breadth-first search, which is called the edge backtracking algorithm.

The edge backtracking algorithm works based on the edge coloring of graphs with predetermined successions of three colors. In this case, when the algorithm finds a conflict in the coloring of the graph, it backtracks to the previous edge, recolors the edge and continues coloring. If there are no possible proper colorings of the problematic edge, the algorithm further backtracks to the edge that precedes both recolored edges. The algorithm continues in this approach until either the whole graph is properly colored or until the algorithm examines all possible edge coloring methods of a given graph.

The time complexity of the edge backtracking algorithm is $O(2^{n-1})$, where n is the number of vertices of a given graph. The algorithm itself is represented in the following steps:

1) The algorithm takes three colors and colors of consecutive edges of graph until either of the following occurs:

- the graph is colored properly,
- or there is conflict in the graph coloring.

2) In the case of conflict in the coloring of graph, the algorithm backtracks to edges that were already colored and recolors them using the next color in predetermined succession of colors until either of the following occurs:
   - the conflict is solved, –where in this case, the algorithm can continue coloring more edges of the graph, as stated in the first step of the algorithm,
   - or all possibilities of edge coloring of the graph are improper, where in this case, the colored graph is snark (cubic graph that cannot be properly colored using three colors).

The algorithm is used in parallel parts of all proposed methods in later sections of this paper.

VI. MODELS OF PARALLEL COMPUTATIONS FOR EDGE COLORING OF GRAPHS

In the examination of graph properties, the edge coloring of the graph is partial operation. Thus, we must reduce the computational time of edge coloring. In [7] and [8], we showed that one graph can be colored at different times if we color it in different orders of edges. In the whole paper, we use the label *permutation of graph* as this changed order of edge coloring. In the algorithms in sections VII - IX, we work with the concept of coloring permuted cubic graphs—graphs with identical structures but different orders of edges. The permutation of graphs is based on a simple matrix multiplication in relation 3.

Let graph G be represented by adjacency matrix A. We must create a large number of permutations of adjacency matrix A; therefore, we create a number of modifications of one graph with different orders of edges.

We use graph automorphism. Let graph G' be automorphic to a given graph G. An important property of automorphic graphs is that any arbitrary pair of such graphs is described by different adjacency matrices but can be presented as the same diagram with different (displaced) labelling of edges and vertices (see Fig. 5). In our case, this means that we work with the same graphs with different orders of vertices and edges.

The permutation of adjacency matrix A of graph G to create an automorphic graph is computed as follows:

$$A' = P^{-1} * A * P \tag{5}$$

where A is the adjacency matrix of given graph G, P is a randomly generated permutation matrix (matrix containing exactly one value 1 in every row and column), P^{-1} is transposed permutation matrix P, and A' is the permuted adjacency matrix of graph G.

The permuted matrix A' obtained by this computation represents the changed order of edges of the original graph G. We use this order (permutation) in our algorithms for edge coloring.

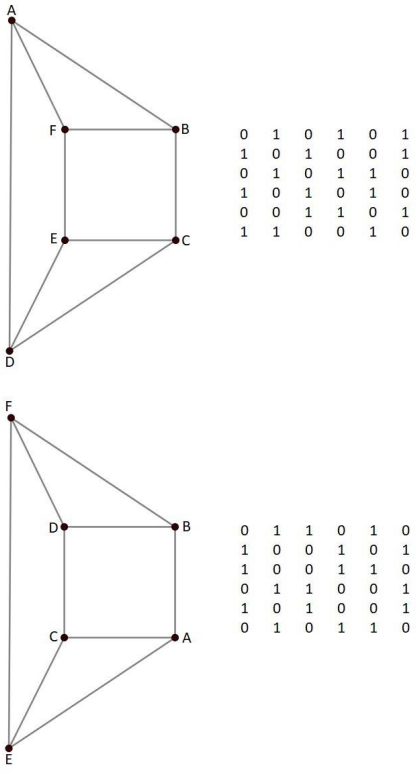

FIGURE 5. Example of isomorphic graphs with their matrices.

In the presented work, we describe the continuation of our research, while we set two objectives, which are explored in the following sections of the article:

- **Computational model based on graph clustering.** The model clusters graphs to groups based on which one of the chosen permutations decreases the computational time of the whole group of graphs.
- **Computational model based on data decomposition.** The model divides the input set of graphs into several parts and colors them in parallel based on principles introduced in Section III.B.
- **Computational model using graph clustering and data decomposition.** This model is a combination of the aforementioned algorithms. It divides the input data set into several parts and clusters the graphs in these parts.

All presented results were computed on part of the high-performance computing system in the High-Performance Computing Center of Matej Bel University in Banská Bystrica [24]. This system contains over 560 processors on 36 nodes.

The data in our experiments were obtained from an online database of graphs called House of Graphs [25]. For the purposes of our research, we chose three sets of 34 vertex snarks. Each set contained 19 935 to slightly over 25 million graphs. The differences among these sets of graphs lay in various properties of the graphs, which are not our focus in this article.

As mentioned in the introduction of the presented paper, we set two main objectives:

Objective 1: Divide a set of graphs to subsets (clusters) so that for each subset of graphs, it is possible to find the order of edge coloring, which reduces the required time to compute each graph in the subset.

We are searching for colorings of graphs that minimize the computational time of edge coloring itself. We chose a constraint of one millisecond, meaning that the algorithm needs to color each graph from the chosen set of graphs in less than one millisecond.

This constraint is important in reducing the time complexity of edge coloring of a set of graphs and from the point of view of a correct decomposition of the problem. The key concept in the decomposition of tasks to subtasks is the similarity (in ideal case equality) of the computational time of all subtasks [18].

In objective 2, we address this significant constraint of the effective decomposition of tasks.

Objective 2: Find such order of edge coloring of graphs that the computational time of edge coloring for all graphs in the set is identical (or within specified tolerance).

VII. ALGORITHM OF FUNCTIONAL DECOMPOSITION BASED ON CLUSTERING

This part of the paper contains an algorithm of functional decomposition based on clustering (AFDC). To examine objective 1 and objective 2, we designed an algorithm of functional decomposition based on clustering.

The input datasets of the algorithm are the following:

- Input set of graphs N stored in graph6 format—a–memory-friendly format to store graphs encrypted in vectors.
- Set of permutations P. We chose a set of 500 permutations computed in previous experiments in [8]. These permutations were chosen based on the computational time of edge coloring, which was the shortest. The chosen permutations are sorted in ascending order of computational time of coloring, where the first permutation corresponds to best (fastest) order of edges for edge coloring.

The algorithms of functional decomposition based on clustering work as follows:

1) Choose the permutation with the lowest time of coloring from set P and label it P_{BEST}. Apply this permutation to the input set of graphs N and edge color this permuted set in parallel using the edge backtracking algorithm.
2) Check the chosen time constraint for every graph in the set. The constraint is designed as follows: Was graph G edge colored in less than one millisecond?
   - TRUE: Graph G is allocated to cluster C_k and removed from set N.
   - FALSE: The algorithm continues with the next step without any action.
3) Permutation P_{BEST} is allocated to cluster C_k and removed from set P.
4) If some graphs remain in set N, the algorithm creates new clusters and cycles back to step 1 of this algorithm. If the set of graphs N is empty, i.e., all graphs were colored in less than one millisecond, the algorithm stops.

The output of the algorithm is a set of k files; –each file contains a cluster of graphs colored in under one millisecond and a permutation, which was applied on each graph in the cluster to reach this time of edge coloring. The algorithm is illustrated by the schema in Fig. 6 and in Algorithm 1.

Algorithm 1 Functional Decomposition Based on Graph Clustering

Require: N, P
  $k = 0$
  repeat
    $k++$
    $P_{BEST} = P_k$

    in parallel do

    use **edgeBacktracking** to color graphs from N while using P_{BEST}
    if $coloringTime(G_n) < 1\ ms$ **then**
      add G_n to cluster C_k
      remove G_n from N
    end if
    remove P_{BEST} from P
    pair P_{BEST} with cluster C_k

    end parallelism

  until N is empty

By implementing this methodology, we expect at least partial fulfillment of objectives 1 and 2. The designed algorithm does not successfully end if there are graphs in N that cannot be colored in the time set under the constraint of the algorithm (one millisecond). A successful run of the algorithm creates k clusters with all graphs from the input set N in them.

In [7] and [8], we show that it is possible to find the order of edge coloring of the graph (permutation), which reduces the computational time of edge coloring for the whole group after it is applied to the entire set of graphs. In Figures 7 and 8, we present a comparison of the maximal computational times of edge coloring for a set of graphs while using randomly generated permutations (Fig. 6) and one verified permutation, which reduces the computational time of edge coloring (Fig. 8).

A. EXPERIMENT 1–EXPERIMENTS ON THE ALGORITHM OF FUNCTIONAL DECOMPOSITION BASED ON CLUSTERING

The presented methodology was experimentally tested on the set of 19 935 snarks with 34 vertices. From the graph in

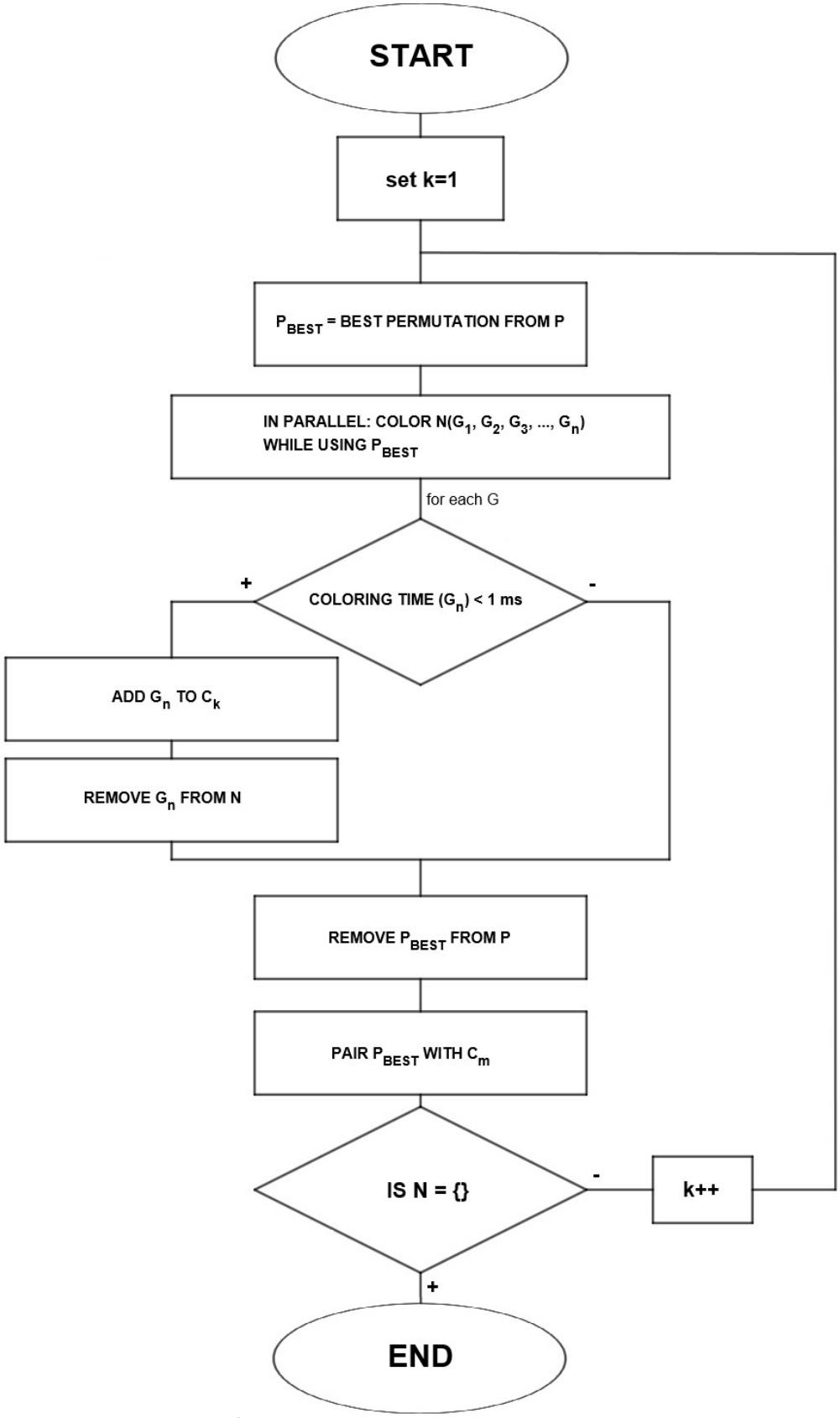

FIGURE 6. Schematic of the algorithm based on graph clustering.

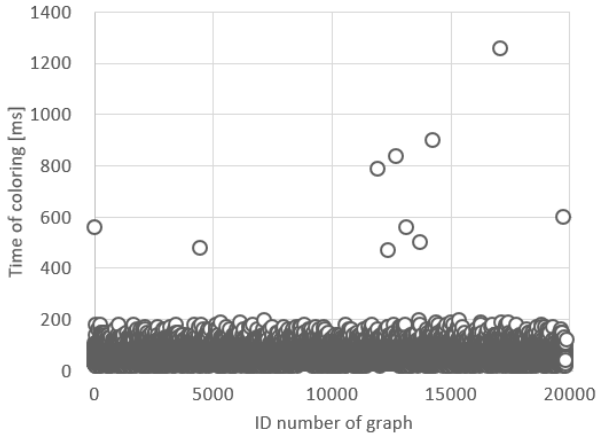

FIGURE 7. Maxima of coloring time for graphs while using random permutations of graphs.

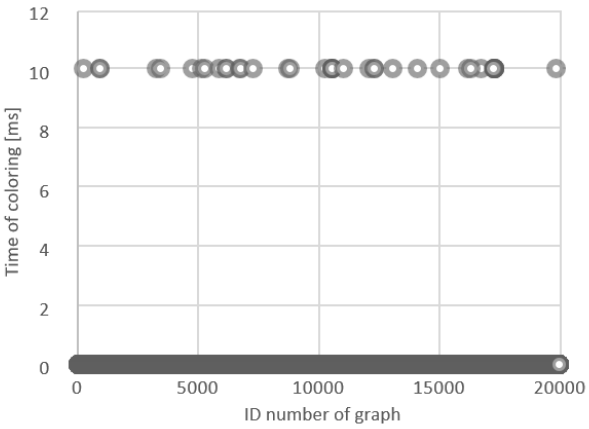

FIGURE 8. Maxima of coloring time for graphs while using chosen permutation with low coloring time.

Fig. 8, this set of graphs clearly contains a large part (near the x-axis), which will be colored in under one millisecond. The set of permutations P was created from the best 500 permutations from previous experiments in [8]. Both input sets N and P were given to the algorithm. The previous experiment clearly shows that when the algorithm successfully runs, the graphs can be divided into 2–40 clusters. The first cluster should contain all graphs that were colored in less than one millisecond (see Fig. 7) using the first permutation in P——this subset of graphs contains 19 896 elements (from the original 19 935). The remaining 39 graphs can be divided into 1–39 clusters based on their edge coloring times after applying other permutations from P.

In this experiment, the algorithm created the following clusters of graphs with their allocated permutations:

- Cluster 1, best permutation $P_{BEST} = \mathrm{P}_1$
  - In total, 19 896 graphs were edge-colored at less than one millisecond.
- Cluster 2, best permutation $P_{BEST} = \mathrm{P}_2$
  - All 39 graphs were edge-colored at less than one millisecond.
- Set N is empty; therefore, the algorithm ends.

TABLE 1. Results of measurements in the experiment on algorithm based on graph clustering.

Computing system	5 nodes 12 processors per node 4 GB of RAM
Problem size	19 935 snarks
Sequential time of coloring [mm:ss]	03:22
Parallel time of coloring [mm:ss]	First cluster: 00:31 Second cluster: 00:02
Speedup of computation	6.122x

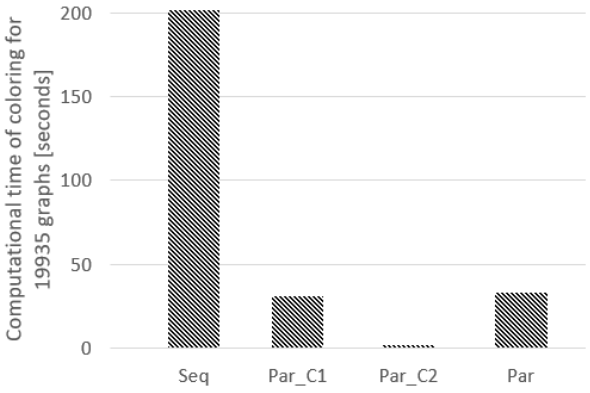

FIGURE 9. Comparison of computing times for edge coloring of the set of 19 935 graphs.

Only two algorithm cycles were required to find clusters with their associated permutations (orders of edge coloring). The results of the measurements of the experiment and the computational system used are listed in Table 1.

In Fig. 9, we present a comparison of computing times of edge coloring for the whole set of 19 935 graphs. We compare sequential computational time of edge coloring (Seq), parallel computational time using proposed algorithm (Par) and required time for edge coloring of graphs in clusters C1 and C2 (Par_C1 and Par_C2).

B. EVALUATION OF EXPERIMENTS ON THE ALGORITHM OF FUNCTIONAL DECOMPOSITION BASED ON CLUSTERING

Experiment 1 shows that it is possible to find a set of permutations of graphs that reduces the computational time of the whole group of graphs to the required limit. In the experiment, we show that to color the chosen set of 19 935 snarks in this manner, the algorithm needs the first two from the chosen 500 permutations (orders of edge coloring). This experiment points to the successful fulfillment of objective 1.

Simultaneously, it is possible to find such an order of edge coloring of the graph that the computational time of coloring of all graphs from the chosen set is similar. With the selection of a suitable permutation algorithm, colored graphs in the set

time (constrained to less than one millisecond). Experiment 1 points to the successful fulfillment of objective 1.

To confirm the results of experiment 1, it is necessary to test the proposed methodology on larger sets of graphs. Thus, we designed and implemented the models in the following subsections of the paper.

VIII. COMPUTATIONAL MODEL BASED ON DATA DECOMPOSITION

To edge color large sets of graphs that contain millions of graphs, we designed and implemented the methodology, which uses suitable permutations of graphs and data processing in batches. This computational model works with a sizable set of graphs in the graph6 format. This dataset is preprocessed by a script, which divides it into subsets (called data parts) in which each subset contains a similar number of graphs. The size of the data part depends on the computing system used; in our case, the size of each data part is at most 500 000.

A. ALGORITHM BASED ON THE DATA DECOMPOSITION MODEL

We designed an algorithm based on a data decomposition model (ADDM). As an **input** for this algorithm, we use a sizable set of graphs in graph6 format, which is divided into data parts. Each data part is distributed over the available computing system and processed by the **algorithm based on the data decomposition model** as follows:

1) Storing data from data part into matrix M. Since graphs are stored in the form of a graph6 vector, it is possible to create a matrix, which can represent the whole set of graphs in a given data part. The number of rows of this matrix is equal to the number of graphs in the data part, and the number of columns is equal to the number of elements in the vector describing graphs in the graph6 format.
2) Parallel edge coloring of graphs. The algorithm processes graphs stored in matrix M in batches. The number of batches is directly dependent on the number of parallel threads, which can be simultaneously run on the computing system:

$$b = n/t \tag{6}$$

where b is the number of batches, n is the number of graphs in the matrix M, and t is the number of parallel threads (see Section *III.B*). The algorithm creates a given number of parallel threads, and each thread obtains one row of matrix M, with one graph in graph6 format. Then, in each parallel thread:

- a graph is translated from graph6 format into the adjacency matrix of the graph,
- the graph is permuted (see relation 3) using permutation with the least coloring time (found in [8]),
- the graph is edge-colored.

After all graphs in the batch are colored, the algorithm joins parallel threads, takes another row of matrix M and repeats this step.

3) After all graphs from the data part are colored, the algorithm records the graph ID and required computational time to edge color the graph into a file, which is shared among all graphs in the data part.

The output of the algorithm consists of several files (the number of files depends on the number of data parts into which the input set of graphs is divided). A schematic of the algorithm is presented in Fig. 9 and Algorithm 2.

Algorithm 2 Algorithm Based on the Data Decomposition Model

Require: N

use **division script** to divide N into n parts

for each data part - concurrently

store data from data part into matrix M
$i = 0, j = 99$
repeat
take graphs (rows of M) from i to $i + j$

in parallel do

use **edgeBacktracking** to color 100 graphs

end parallelism

$i = i + j + 1$
until all graphs edge colored

B. EXPERIMENT 2–EXPERIMENTS ON THE ALGORITHM BASED ON THE DATA DECOMPOSITION MODEL

The proposed algorithm was experimentally tested on the set of graph data, the sequential computation time of which was too high to use sequential algorithms for their coloring. This dataset contains 3 833 587 snarks with 34 vertices. The graph set was divided into eight similar data parts (approximately 479 198 graphs per part). The data parts were colored in batches of 100 graphs. The measurement results and comparison of the proposed parallel method and sequential method in terms of the required computational time and memory are presented in Table 2 and Fig. 11. In Table 2, sequential algorithms are labeled *Seq*, and the presented algorithm is labeled with abbreviation *ADDM* (algorithms based on the data decomposition model).

From Table 2 and Fig. 11, the proposed algorithm based on the data decomposition model significantly speeds up the computations containing millions of graphs, which dramatically increases the required memory for computation of the algorithm. The computing system, which contains 5 nodes and 12 processors per node, is based on a virtualization

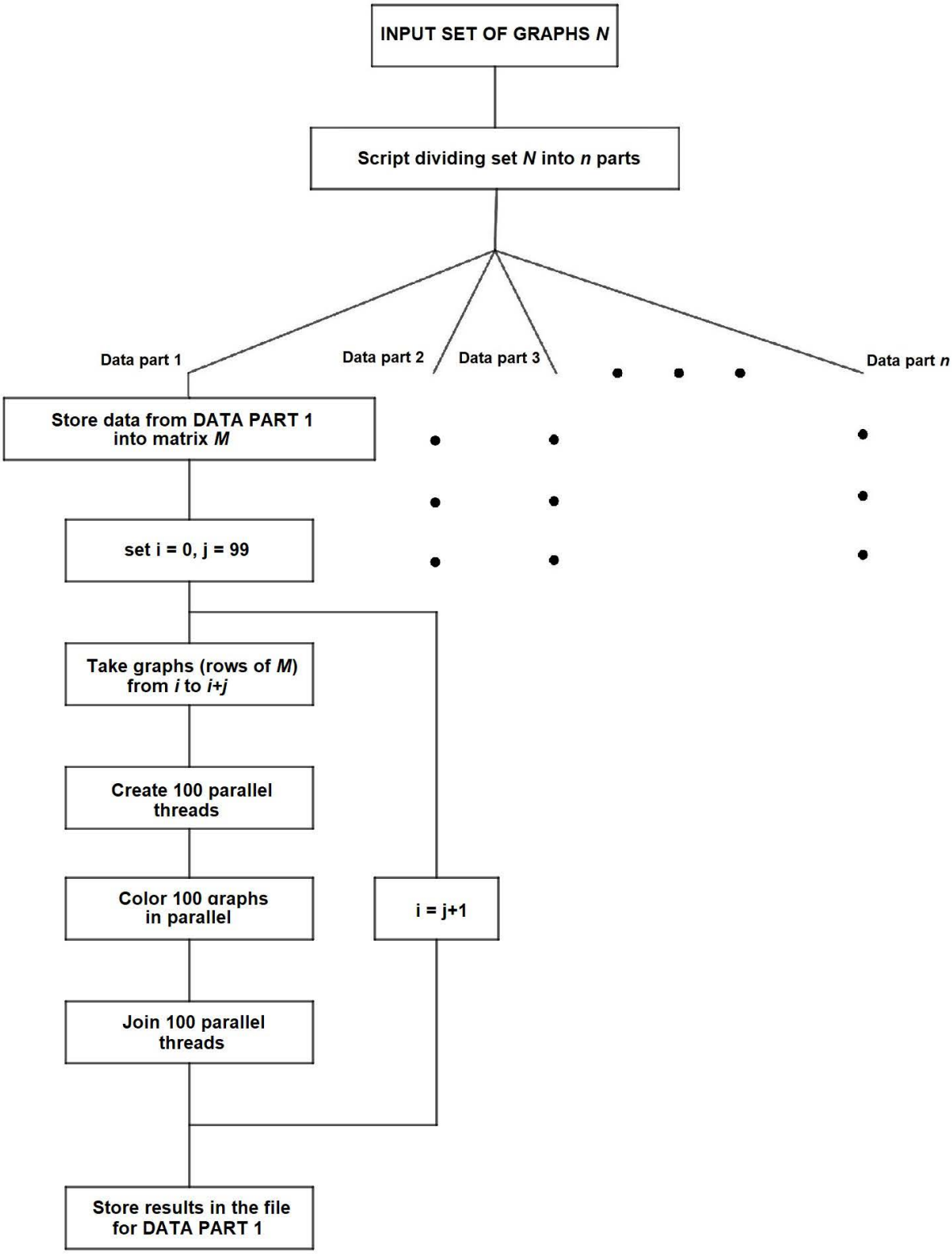

FIGURE 10. Schematic of the algorithm based on the data decomposition model.

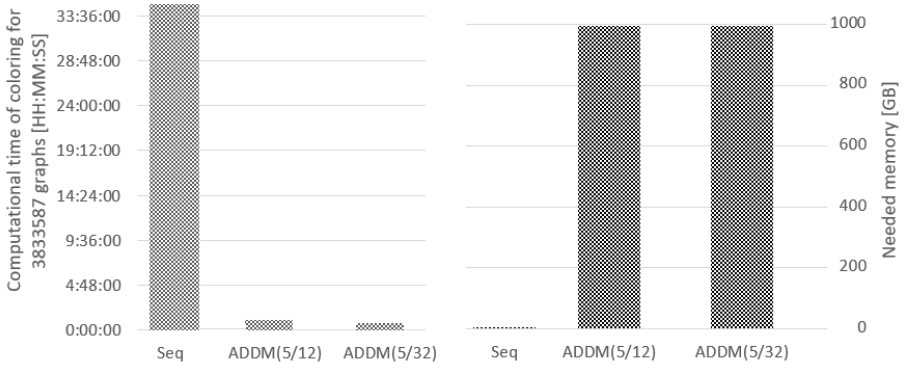

FIGURE 11. Comparison of the required time and memory consumption.

TABLE 2. Comparison of the required time and memory consumption.

	Seq	ADDM1	ADDM2
Problem size	3 833 587	3 833 587	3 833 587
Computing system (nodes/processors per node)	-	5/12	5/32
Computation time [hh:mm:ss]	34:47:53	01:06:58	00:42:14
Memory needed [kB]	3 450 900	994 598 552	993 818 164
Speedup of computation	N/A	31.178x	49.437x
Memory requirement growth	N/A	288.116x	287.988x

on the High-Performance Computing Cluster at Matej Bel University in Banská Bystrica, Slovakia.

C. EVALUATION OF EXPERIMENTS ON ALGORITHMS BASED ON A DATA DECOMPOSITION MODEL

Compared to the sequential algorithm for the edge coloring set of graphs, the parallel/distributed approach in this section of the paper reached a parallel speedup of 31.2, while the required RAM for computation was approximately 288.1 times higher. A detailed analysis of the results shows that from the input set of 3 833 587 graphs:

- 3 301 223 graphs were edge-colored in less than one millisecond,
- 532 364 graphs were colored in more than one millisecond, while:
  - 484 267 graphs were colored in 10–20 milliseconds,
  - 38 486 graphs were colored in 20–30 milliseconds,
  - 6 881 graphs were colored in 30–40 milliseconds,
  - 1 750 graphs were colored in 40–50 milliseconds,
  - 735 graphs were colored in 50–60 milliseconds,
  - 175 graphs were colored in 60–70 milliseconds,
  - 70 graphs were colored in 70–80 milliseconds.

This analysis of the results points us toward combining the above algorithms to satisfy both set objectives of this paper.

IX. COMPUTATIONAL MODEL USING GRAPH CLUSTERING AND DATA DECOMPOSITION

From the measurements in the previous section, the algorithm based on the data decomposition model is clearly a substantial improvement in terms of reducing the computational time of edge coloring of graph sets. The achieved result is not satisfying in terms of objective 2, which requires similar computational times of coloring for all graphs in the colored set. From the results of experiment 2, the chosen set of graphs was colored in 0.2–80 milliseconds.

A. ALGORITHM USING GRAPH CLUSTERING AND DATA DECOMPOSITION

We designed the following combination of algorithms, which secures a decrease in time to color the graph set and satisfies the condition of objective 2 (the time constraint is, again, one millisecond). As an **input** for the algorithm, we use the same sets as in the algorithm based on graph clustering - set of graphs N and set of permutations P.

The algorithm using graph clustering and data decomposition (ACDD) is described as follows:

1) Division of graph set N into data parts. Similar to the algorithm based on the data decomposition model, the algorithm divides the input set of graphs N into n smaller parts using a division script. Specifically, the script divides the input dataset into approximately equal subsets of approximately 500 000 elements.
2) Distribution of data parts. Based on the data decomposition model algorithm, the data parts (subsets) created in the previous step are distributed over the available computing system.
3) In the next step, the proposed algorithm executes individual steps of the algorithm based on graph clustering for each of n data parts created from input dataset N:
   - Choose the permutation with the lowest time of coloring P_{BEST} from P and apply it on the entire data part.
   - Edge color the graphs in the data part in parallel using the parallel thread model.
   - Check the constraint for all graphs in the data part: Was graph G edge colored in less than one millisecond?
   - Create cluster of graphs that satisfy the above condition.

- Allocate P_{BEST} to the created cluster of graphs.
- Remove P_{BEST} from set P and remove the graphs with coloring time under one millisecond from the data part.

These steps are repeated until the given data part is not empty or the algorithm has unsuccessfully attempted all permutations (e.g., there is a graph that cannot be colored in less than one millisecond).

Algorithm 3 Algorithm Using Graph Clustering and Data Decomposition

Require: N, P

use **division script** to divide N into n parts - $N_1, N_2, \ldots, N_n$

for each data part - concurrently

$k = 0$
repeat
$k++$
$P_{BEST} = P_k$

in parallel do

use **edgeBacktracking** to color graphs from N while using P_{BEST}
if $coloringTime(G_n) < 1\ ms$ **then**
add G_n to cluster C_k
remove G_n from N
end if
remove P_{BEST} from P
pair P_{BEST} with cluster C_k

end parallelism

until data part is empty

The output of the algorithms consists of a set of files containing clusters of graphs with their allocated permutations for each of n data parts. To collect results, it is necessary to use an agglomeration function on the set of files. This function connects files (clusters) with identical permutations.

A schematic of the algorithm using graph clustering and data decomposition is presented in Fig. 12 and Algorithm 3.

B. EXPERIMENT 3–EXPERIMENTS ON ALGORITHM USING GRAPH CLUSTERING AND DATA DECOMPOSITION

The algorithm using graph clustering and data decomposition was experimentally tested on all available data sets for snarks with 34 or more vertices [25]:

- set of 3 833 587 graphs, which contains snarks with 34 vertices,
- set of 25 286 953 graphs, which contains snarks with 34 vertices,
- set of 180 612 graphs, which contains snarks with 36 vertices,
- set of 60 167 732 graphs, which contains snarks with 36 vertices,
- set of 404 899 916 graphs, which contains snarks with 36 vertices,
- set of 35 429 graphs, which contains snarks with 38 vertices,
- set of 39 graphs, which contains snarks with 38 vertices,
- set of 19 775 768 graphs, which contains snarks with 38 vertices,
- set of 25 graphs, which contains snarks with 40 vertices.

TABLE 3. Graph clusters for the set of 3 833 587 snarks with 34 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	01:06:58	994.598	532 364
2	2	00:07:00	32.059	57 274
3	3	00:00:49	1.457	5 775
4	4	00:00:07	0.254	0

TABLE 4. Graph clusters for the set of 25 286 953 graphs with 34 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	04:19:09	3219.341	1 356 175
2	2	00:11:43	12.242	127 775
3	3	00:01:26	3.362	15 032
4	4	00:00:16	0.845	6
5	5	00:00:01	0.017	0

TABLE 5. Graph clusters for the set of 180 612 graphs with 36 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	00:01:10	1.326	4008
2	2	00:00:07	0.839	241
3	3	00:00:07	0.376	12
4	4	00:00:01	0.017	0

The results of the measurements from this experiment are presented in Tables 3-11. All values in the tables were measured on the computing system consisting of 5 nodes and 12 processors per node in the HPCC UMB in Banská Bystrica, Slovakia.

From the previous experiments (mainly experiment 2), the algorithm based on the data decomposition model has high memory requirements. In addition to the considerable memory requirements of the presented algorithm, it is necessary to show other costs of computation - the time required to decompose the input data set.

The time required for a script which takes care of the decomposition of an input dataset (as described in section *VIII.A*) is linear - $O(n)$, where n is the size of the data part (in our case 500 000 graphs). This time is negligible compared to computational time of the whole task when using smaller datasets.

In the case of largest used data sets, the algorithm needed:

- almost 3.2 TB of RAM for the data set of 19 775 768 snarks with 38 vertices, the time for decomposition of the dataset was 4 minutes 34.1 seconds,

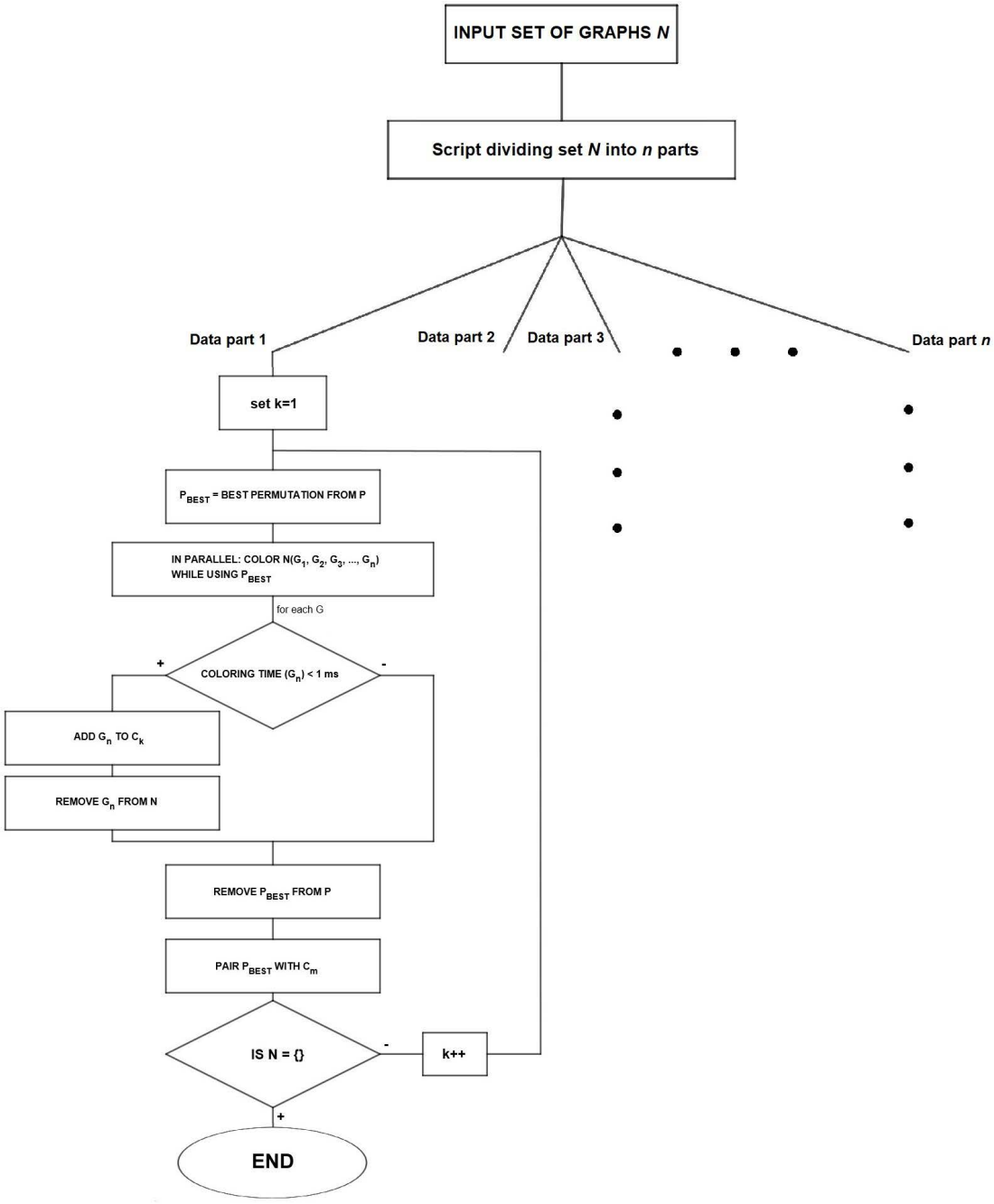

FIGURE 12. Schematic of the algorithm using graph clustering and data decomposition.

- almost 4 TB of RAM for the data set of 25 286 953 snarks with 34 vertices, the time for decomposition of the dataset was 7 minutes 40 seconds,
- almost 4.5 TB of RAM for the data set of 60 167 732 snarks with 36 vertices, the time for decomposition of the dataset was 16 minutes 43.2 seconds,
- almost 9 TB of RAM for the data set of 404 899 916 snarks with 36 vertices, the time for decomposition of the dataset was 1 hour 24 minutes and 32.4 seconds.

Although in these cases the overheads are not small, together with the computational time of the problem, our approaches still represent a considerable speedup of the computation itself. In the tables containing outcome of experiments, we present the measurement of the following properties of computation:

- Cluster number: ID number of cluster that contains clustered graph data,
- P_{BEST} number: ID number of permutations used in a given cluster,
- Computing time: time to color all graphs in a given cluster,
- Memory needed: RAM needed to run the program,

TABLE 6. Graph clusters for the set of 60 167 732 graphs with 36 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	08:21:18	4372.881	4 229 360
2	2	00:34:31	35.237	256 902
3	3	00:03:11	1.245	53 487
4	4	00:01:07	0.891	16 343
5	5	00:00:33	0.873	8 901
6	6	00:00:31	0.754	8 042
7	7	00:00:04	0.019	0

TABLE 7. Graph clusters for the set of 404 899 916 graphs with 36 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	47:16:14	8472.230	17 642 937
2	2	03:32:49	56.671	12 876 022
3	3	00:56:11	30.290	4 038 765
4	4	00:40:26	12.128	1 844 397
5	5	00:38:39	9.133	1 000 036
6	6	00:07:12	2.465	321 009
7	7	00:05:31	1.026	162 932
8	8	00:00:22	0.891	21 329
9	9	00:00:20	0.806	17 890
10	10	00:00:18	0.650	12 993
11	11	00:00:17	0.521	9 636
12	12	00:00:13	0.360	4 109
13	13	00:00:10	0.332	1 572
14	14	00:00:03	0.196	60
15	15	00:00:02	0.183	38
16	16	00:00:03	0.067	0

TABLE 8. Graph clusters for the set of 35 429 graphs with 38 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	00:00:42	0.630	1053
2	2	00:00:11	0.439	0

TABLE 9. Graph clusters for the set of 39 graphs with 38 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	00:00:03	0.371	0

TABLE 10. Graph clusters for the set of 19 775 768 graphs with 38 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	03:16:02	3018.03	594 945
2	2	00:01:33	1.139	10 709
3	3	00:00:03	0.732	4 283
4	5	00:00:01	0.424	0

- Number of graphs colored in more than one millisecond. These graphs must be recolored using another permutation.

C. EVALUATION OF EXPERIMENTS ON THE ALGORITHM USING GRAPH CLUSTERING AND DATA DECOMPOSITION

We can compute the required time to color the whole set of graphs while considering the condition of objective 2 as follows:

$$T_o = \sum_{i=1}^{k} Tc_{max_i} \tag{7}$$

where T_o is the overall computational time to edge color a given set of graphs, k is the number of clusters, and Tc_{max_i} is the computational time of cluster number i.

In the experiments that used the presented computational model, we worked with data sets of different sizes. In addition to the set sizes, the individual data sets differed in the number of vertices of the graphs (34, 36, 38 and 40) they contained and other mathematical properties (girth and cyclic connectivity of the graph) - although we do not work directly with these properties as in [26].

From Table 3, for the problem of edge coloring of the graph set, which contains 3 833 587 graphs, we needed four clusters, which were computed in 01:14:54. Therefore, we reached the time of coloring for each graph in the set, which was lower than one millisecond using four orders of edge coloring (permutations).

Table 4 shows that for the set of 25 286 953 graphs, we needed five clusters (five permutations) to reach the limit of coloring time for each graph in the set. This computation took 04:32:35.

Similar to 34-vertex graphs, we proceeded with three sets of 36-vertex graphs. The first subset contains 180 612 graphs and the total computation time of its edge-coloring was 00:01:25. The other two subsets of 36-vertex snarks contain more than 60 million and more than 404 million graphs. The total computation time for the set of 60 167 732 graphs was 09:01:15. For the set of 404 899 916 snarks, the time of coloring computation came up to 53:18:50. There is a fourth set of graphs in the section of 36-vertex snarks, but it contains only one element and therefore is not fitting for our experiments.

When working with 38-vertex snarks we used three sets of graphs. The first of them contained 35 429 elements and the presented algorithm was able to edge-color it in 53 seconds. The second set of 38-vertex snarks was a small set of 39 graphs that were colored in 3 seconds. The edge-coloring of last subset of 38-vertex snarks, which contains 19 775 768 graphs, was computed in 03:17:39.

In the case of 40-vertex snarks, we have only one set of 25 graphs, which, however, was surprisingly divided into three clusters and colored in 6 seconds.

In all experiments (where it was possible to make a comparison), it was confirmed that the use of our computation method leads to a significant reduction in computation time.

TABLE 11. Graph clusters for the set of 25 graphs with 40 vertices.

Cluster number	P_{BEST} number	Computing time [hh:mm:ss]	Memory needed [GB]	Number of graphs colored >1 ms
1	1	00:00:04	0.230	9
2	2	00:00:01	0.089	2
3	3	00:00:01	0.019	0

TABLE 12. Comparison of sequential and parallel computational times and number of permutations on 34-vertex snarks.

Number of graphs	Sequential time [hh:mm:ss]	Parallel time [hh:mm:ss]	Speedup	Number of permutations
19 935	00:03:22	00:00:33	6.122x	2
3 833 587	34:47:53	01:14:54	31.178x	4
28 286 953	N/A	04:32:35	N/A	5

For example, for 3 833 587 graphs, the original sequential computation took approximately 34 hours and 48 minutes, and after using our method, the same computation took 1 hour and 14 minutes - indicating approximately 31-fold acceleration of the computation (see Table 12). Using our method, it was possible to make a computation for large sets (approximately 60 and 404 million) of graphs with 36 vertices, for which it was not practical to perform sequential computations.

Although the algorithm needs a very large memory, it allows decent parallel speedup of computation and can create clusters of graphs that have assigned their permutations, which can satisfactorily color graphs from the viewpoint of objective 2 (consequently from the viewpoint of effective decomposition).

X. CONCLUSION

In this paper, we have established the basic criteria for the correct decomposition of computations in HPC systems. These criteria are based on load balancing and time balancing concepts as presented in the section *III* and were used in design of our main objectives:

- Divide a set of graphs to subsets (clusters) so that for each subset of graphs, it is possible to find the order of edge coloring, which reduces the required time to compute each graph in the subset.
- Find such order of edge coloring of graphs that the computational time of edge coloring for all graphs in the set is identical (or within specified tolerance).

In order to explore set objectives, we have designed and implemented models of computation that can be used to find a set of such permutations that all graphs in a given set require less time to color. All presented experiments confirm significant reduction of computation time needed for edge coloring of large sets of graphs.

All measurements in this paper (see Tables 3-11) show that when different orders of graph coloring are used, we require low number of permutations (e.g. for approximately 25 million graphs we need 5 clusters, for approximately 404 million graphs we need 16 clusters), while for each cluster algorithms uses one order of edge coloring of graphs.

Although the parallel speedup of the computation is considerable, the disadvantage of the parallel approaches is the memory requirements (as can be seen from Tables 3-11).

Generally, the memory consumption of the algorithm and the number of parallel threads that can run simultaneously depends on the system we use to compute the problem. When working with the proposed computational models, it is necessary to take into account the basic thesis of parallel computation [28], [29], which says that the computational time required to solve a problem using a parallel algorithm is polynomially related to memory space required for sequential computation of the same problem. As can be seen in section *IX.B*, the memory requirements for proposed algorithms are relatively high - based on used dataset it varies from 3.2 to 9 TB of RAM.

The number of parallel parts for the task of edge coloring complexity depends mainly on the memory possibilities of the system. In our case, however, we have a system with a sufficiently high memory capacity and therefore it is possible to process a large number of subtasks in parallel. During the experiments, we encountered a low-memory problem several times, which was simply fixed by reducing the number of parallel threads in the computation. The maximal number of parallel parts which could be used in our high performance computing system was 500. This number of parallel parts was determined mostly by experimentation and can be different on any other system.

Future works related to the presented results can focus on examining several main areas of research.

Graph coloring is usable in various areas of research, –including scheduling, radio frequency allocation, compiler optimization and SAT solvers. It would be fitting to design and implement methodology to compare the proposed algorithms and models in this article and the standard solution models for the aforementioned problems.

It is possible to search for properties that define the order of edge coloring of the graph (permutation) as low (resp. high) computational time. If such properties are distinguishable, it is possible and necessary to create an algorithm to generate such permutations for various types and sizes of graphs. Also, common properties of graphs in created clusters need to be examined and analyzed.

Other than working directly with known properties of graphs, implementation of methods of artificial intelligence similar to [27], which would be able to analyze graph coloring and propose fitting permutations for given set of graphs is desirable.

REFERENCES

[1] D. Marx, "Graph colouring problems and their applications in scheduling," *Periodica Polytechnica Elect. Eng.*, vol. 48, nos. 1-2, pp. 11–16, 2004.
[2] G. J. Chaitin, "Register allocation & spilling via graph coloring," in *Proc. SIGPLAN Symp. Compiler Construct. (SIGPLAN)*, 1982, pp. 98–105.
[3] L. Kowalik, "Improved edge-coloring with three colors," *Theor. Comput. Sci.*, vol. 410, nos. 38–40, pp. 3733–3742, Sep. 2009.
[4] R. Beigel and D. Eppstein, "3-coloring in time O(1.3289^n)," *J. Algorithms*, vol. 54, no. 2, pp. 168–204, 2005.
[5] M. A.–Vilaltella. J. Fiol, "A simple and fast heuristic algorithm for edge-coloring of graphs," *AKCE Int. J. Graphs Combinatorics*, vol. 10, no. 3, pp. 263–272, 2013.
[6] J. Karabas, E. Macajova, and R. Nedela, "6-decomposition of snarks," *Eur. J. Combinatorics*, vol. 34, no. 1, pp. 111–122, 2013.
[7] A. Dudas, P. Vostinar, J. Skrinarova, and J. Silaci, "Improved process of running tasks in the high performance computing system," in *Proc. 16th Int. Conf. Emerg. eLearning Technol. Appl. (ICETA)*, Nov. 2018, pp. 133–140.

[8] A. Dudas, J. Skrinarova, and E. Vesel, "Optimization design for parallel coloring of a set of graphs in the high-performance computing," in *Proc. IEEE 15th Int. Sci. Conf. Informat.*, Nov. 2019, pp. 93–99.
[9] A. Dudas and J. A. S. Kiss, "On graph coloring analysis through visualization," in *Proc. Inf. Digit. Technol. Int. Conf.*, Jun. 2021, pp. 71–78.
[10] R. Ding, R. Zhao, and L. Han, "An automatic computation and data decomposition algorithm of prioritized dominant array," in *Proc. 13th Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, Dec. 2012, pp. 305–308.
[11] D. Rui, Z. Rongcai, and L. Xiaoxian, "An improvement to affine decomposition on distributed memory architecture," in *Proc. 11th Int. Symp. Distrib. Comput. Appl. Bus., Eng. Sci.*, Oct. 2012, pp. 18–21.
[12] H.-J. Li, Z. Wang, J. Pei, J. Cao, and Y. Shi, "Optimal estimation of low-rank factors via feature level data fusion of multiplex signal systems," *IEEE Trans. Knowl. Data Eng.*, early access, Aug. 13, 2020, doi: 10.1109/TKDE.2020.3015914.
[13] H. J. Li, Z. Bu, Z. Wang, and J. Cao, "Dynamical clustering in electronic commerce systems via optimization and leadership expansion," *IEEE Trans. Ind. Informat.*, vol. 16, no. 8, pp. 5327–5334, Aug. 2020.
[14] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL, USA: CRC Press, 2010.
[15] J. Skrinarova and M. Povinsky, "Parallel simulation of tasks scheduling and scheduling criteria in high-performance computing systems," *J. Inf. Organizational Sci. Varaždin Univ. Zagreb, Fac. Org. Inform.*, vol. 43, no. 2, pp. 211–228, 2019.
[16] I. Holyer, "The NP-completeness of edge-colouring," *SIAM J. Comput.*, vol. 10, no. 4, pp. 718–720, 1981.
[17] D. Hentrich, E. Oruklu, and J. Saniie, "A dataflow processor as the basis of a tiled polymorphic computing architecture with fine-grain instruction migration," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 10, pp. 2164–2175, Oct. 2018.
[18] J. Kollár, *Methods and Tools for Effective Parallel Computations* (Edition of Monographies FEI TU in Košice). Košice, Slovakia: Academic (in Slovak), 2003.
[19] J. Kollar and M. Vagac, "Aspect-oriented approach to metamodel abstraction," *Comput. Informat.*, vol. 31, pp. 983–1002, Nov. 2012.
[20] A.-Gupta, A. Karypis, G. Grama, and V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Boston, MA, USA: Addison Wesley, 2003.
[21] S. Palúch and Š. Peško, *Quantitative Methods in Logistics*, (*in Slovak*). Zilina, Slovakia: EDIS, 2006.
[22] R. Diestel, *Graph Theory*. Berlin, Germany: Springer, 2016.
[23] J. Hägglund, "On snarks that are far from being 3-edge colorable," *Electron. J. Combinatorics*, vol. 23, no. 2, p. P2, Apr. 2016.
[24] *High Performance Computing Center: High Performance Computing Center*. Accessed: Jun. 20, 2021. [Online]. Available: http://www.hpcc.umb.sk
[25] G. Brinkmann, K. Coolsaet, J. Goedgebeur, and H. Mélot, "House of graphs: A database of interesting graphs," *Discrete Appl. Math.*, vol. 161, nos. 1–2, pp. 311–314, Jan. 2013. [Online]. Available: http://hog.grinvin.org
[26] M. Binu, S. Mathew, and J. N. Mordeson, "Cyclic connectivity index of fuzzy graphs," *IEEE Trans. Fuzzy Syst.*, vol. 29, no. 6, pp. 1340–1349, Jun. 2021.
[27] Y. Xiong, Y. Zhang, X. Kong, H. Chen, and Y. Zhu, "GraphInception: Convolutional neural networks for collective classification in heterogeneous information networks," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 5, pp. 1960–1972, May 2021.
[28] T. Pinchinat, S. Schwarzentruber, and F. Charrier, "Model checking against arbitrary public announcement logic: A first-order-logic prover approach for the existential fragment," in *Dynamic Logic. New Trends and Applications. DALI 2017* (Lecture Notes in Computer Science). vol. 10669. Cham, Switzerland: Springer, pp. 133–152.
[29] A. K. Stockmeyer and L. J. Chandra, "Alternation," in *Proc. 17th Annu. Symp. Found. Comput. Sci. (SFCS)*, 1976, pp. 98–108.

JARMILA SKRINAROVA was born in Zlín, Czech Republic, in 1962. She received the M.S. degree in automation and management and the Ph.D. degree in technical cybernetics from Slovak Technical University, Slovakia, in 1986 and 2004, respectively.

Since 2013, she has been an Associate Professor with the Department of Computer Science, Faculty of Natural Sciences, Matej Bel University, Banská Bystrica, Slovakia. She is the author or coauthor of 95 research articles, educational texts, and technical books. Her research interests include parallel and distributed computing, and optimization and scheduling of tasks in systems.

ADAM DUDÁŠ was born in Banská Bystrica, Slovakia, in 1992. He received the B.S. and M.S. degrees in computer science from Matej Bel University, Banská Bystrica, in 2015 and 2017, respectively, and the Ph.D. degree in computer science from the University of Zilina, Slovakia, in 2020.

Since 2020, he has been an Assistant Professor with the Department of Computer Science, Faculty of Natural Sciences, Matej Bel University. He is the author or coauthor of 18 research articles. His research interests include parallel and distributed computing, optimization of graph coloring, and big and irregular data and databases.