Heterogeneous Computational Scheduling using Adaptive Neural Network

Alexander Allahverdyan[1], Anastasia Zhadan[1], Ivan Kondratov[1], Ovanes Petrosian[1]*, Aleksei Romanovskii[2] and Vitaliy Kharin[2]

[1] Saint-Petersburg State University, Saint-Petersburg, 198504, Russia.

[2] Huawei, Saint-Petersburg Research Institute, Saint-Petersburg, 191119, Russia.

Contributing authors: aleka_alexander@mail.ru; anastasiya_markel@mail.ru; kondratov.ivan.vladimirovich@gmail.com; petrosian.ovanes@yandex.ru; aleksei.romanovskii@huawei.com; vitaliy.kharin@huawei.com;

Abstract

Graph scheduling is important for improving system performance in the distributed heterogeneous computing environment. Since in many areas computational workflow can be represented by a Directed Acyclic Graph (DAG) e.g. operating systems, data engineering, machine learning. Due to its key importance, the DAG tasks scheduling problem has been extensively studied in the literature and a large number of researches by major it companies Microsoft[33],[34], Amazon[35], Google[36] and IBM[37] focus on scheduling apps/workflows modeled as static DAGs in various application areas. As a result many state-of-the-art heuristic algorithms such as DONF, CPOP, HCPT, HPS and PETS have been proposed however that leave a lot of space for optimization that needs to be explored. Therefore, the main goal of this research is propose a scheduling scheme based on Artificial Neural Networks (ANN) outperforming state-of-the-art DAG scheduling algorithms (to reduce the DAG makespan) suitable on small-scale and large-scale graphs and adaptable

*corresponding author

to different workspaces. Set of graph metrics – the numerical representation of each DAG node on a local scale, which chosen based on recent research in the field of DAG scheduling [1],[2], are used as input to the neural network, while the output is the rank of the estimated node, or its priority of execution. The novelty of our work is supported by the proposed algorithm architecture based on the concept of an adaptive DAG scheduling metric, where state-of-the-art graph scheduling metrics are aggregated by the neural network. During training and selection of weights, the neural network actually becomes a new metric that adapts to the topology of the graph, which allows for more efficient scheduling for DAG in terms of reducing makespan. In order to prove the efficiency of the approach a comparison with the state-of-the-art DAG scheduling algorithms is provided: DONF, CPOP, HCPT, HPS and PETS. Based on the simulation results the proposed algorithm shows the improvement of up to 39% on specific graph topologies, and average performance gain of 6.7% compared to the best scheduling algorithm on large number of random small-scale DAGs. Another important area of research is the determination of the available optimization space, for which the problem is formulated in terms of Mixed-Integer Linear Programming (MILP) and solved for a set of small-scale graphs(computationally heavy, not suitable for the large scale scheduling). On large-dimensional graphs obtaining global optimal solution is possible, but the complexity grows more than quadratic, so to solve real applied problems this approach will require huge computational resources, and in some cases is simply not feasible. For example, in cloud computing, the structure and topology of a graph change at high speed, so a faster algorithm for obtaining task schedules is needed. Therefore, we propose an algorithm based on adaptive heuristics that based on testing covers 39.2% of the proximity interval from the best scheduling algorithm to the global optimal solution obtained using the MILP approach. And also the results of testing on a set of large-scale dimensions DAGs are given, where the average performance gain vary from 6.51% to 31,43% compared to the best heuristic algorithm depending on the workspace and the dimensionality of the graph. This suggests that the idea of using artificial intelligence and neural networks methods for graph scheduling shows good results and requires further research.

Keywords: Neural networks, Scheduling , Directed Acyclic Graph, Genetic Algorithm

1 Introduction

Modern computing systems often use heterogeneous computing systems. These systems usually have several computing units of a different type. One of the key factor in achieving high performance in modern computing systems is the efficient resource scheduling. The general approach for task scheduling is divided into two phases [3], sorting the tasks and assigning the tasks to the appropriate executors for processing. The scheduling problem is more complicated

for dependent tasks, such as scientific calculations and big data applications, which have operational dependencies between their different parts. The workflow represented as data dependencies between the tasks by the nodes edges. The tasks in the workflow are represented by the nodes in the graph. Static DAG scheduling is the well-known problem where it is necessary to to schedule all tasks of a complex application to the appropriate executor and to minimize the total execution time. In general, finding the optimal solution to the stated problem is an NP-complete problem [4–6].

A large number of studies of major it companies focus on scheduling apps/-workflows modeled as DAGs in various application areas. Cloud providers, such as Microsoft [33], [34], Amazon [35], Google [36] and IBM [37] have publicly available research on process optimization and scheduling. Distributed application frameworks, such as Spark [38] or Flink [39] use directed acyclic graphs (DAGs) to represent applications and different approaches to scheduling. It is important to notice that, as in our study the workflows are represented as static DAGs, and tasks in DAGs execute different functions on different data with varying resource demands and task runtimes. In the work of researchers from Google [40] a distributed machine learning frameworks, such as TensorFlow, have been extensively explored in optimizing the learning process and inference.

This article will discuss static scheduling models, where the information of the computational graph (adjacency matrix, weights of nodes and edges) is known before running the compilation. Static scheduling algorithms are further categorized in the following way: mixed integer programming approach [7, 8], heuristic-based algorithms [9–17], machine learning methods [18–24, 29–32, 41].

Mixed integer programming approach (MILP). On the first step the DAG scheduling problem is formulated as a MILP and on the second step one of the state-of-art MILP solvers such as Gubori [25], Cplex [27], CBC [26] or others can be used to solve it. The MILP approach provides a global optimal solution to the stated DAG scheduling problem, but unfortunately it is intractable for most practical-scale problems due to the high dimensionality and computational complexity. However in this research we are using the MILP approach to study of how far the proposed algorithm is from the global optimal solution.

Heuristic-based algorithms. Although these heuristic algorithms have good versatility and stability [9] and have polynomial time complexity instead of exponential time complexity. Scheduling strategies usually are based on greedy strategies and therefore there is still much room for improvement. The classification of heuristic algorithms for scheduling tasks on graphs can be found on the Figure 1. Heuristics based list task-scheduling algorithms that can be executed in heterogeneous processors environment: Levelized-Min Time (LMT) [28], Dynamic Level Scheduling (DLS)[10], Critical Path on a Processor (CPOP)[11], Mapping Heuristic (MH)[12], High Performance Task Scheduling (HPS) [13], Heterogeneous Earliest Finish Time (HEFT) [14],

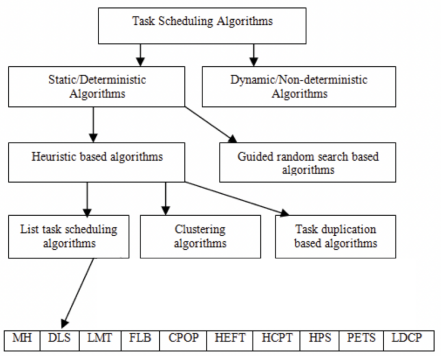


Fig. 1: Categorization of DAG scheduling algorithms[28]

Heterogeneous Critical Parent Tree (HCPT)[28], Performance Effective Task Scheduling (PETS) [15]), Look ahead [16], and Degree of Node First (DONF) [17]. According to the results in [28] and [17] the most effective and the frequently used algorithms for static DAG scheduling (in the problem of minizing the makespan) are the algorithms CPOP, HCPT, HPS, PETS and DONF. These algorithms were selected for comparison with the algorithm proposed in this research.

Machine learning methods. As a prime example of using machine learning for graph scheduling is Reinforcement Learning (RL)[29] and Monte Carlo Tree Search (MCTS)[30] which can learn from past scheduling strategies and adjust current actions to optimize scheduling results. The recently proposed task scheduling algorithm that are based on reinforcement learning either greatly simplifies the scheduling model or requires a large amount of computing resources for training. Orhan [31] proposed a heterogeneous distributed system scheduling method based on reinforcement learning. But some assumptions about this approach, such as task type, machine performance, and cluster state, do not quite match real DAG tasks scheduling. Loth M [32] proposed using Monte Carlo Tree Search (MCTS) to solve job shop scheduling problems. The goal of job shop scheduling is to determine the processing sequence of the operations on each machine and the start time of each operation. However, compared with DAG task scheduling, its data dependency is more simplified.

In this paper an adaptive metric neural network based approach for DAG scheduling is proposed for the first time. Figure 2 shows the overall structure of the proposed algorithm. The first level of the algorithm is the neural network that determines the execution queue of the nodes according to the characteristics of the nodes. The second level of the algorithm is the assignment of a task to a computing resource of the same type according to the Earliest Finish Time (EFT) algorithm. It is important to notice that the input features of vertices defined by the set of state-of-the-art graph metrics (section 4.2) that determine the priority for each task from the set of ready to perform tasks.

A genetic algorithm [41] is used to train a neural network [42]. Genetic algorithms are used for exploring a large and complex space in an intelligent way to find values close to the global optimum. The approach of adaptive metric neural network allows to avoid bad performance of AI learning based algorithms due to the fact that the proposed algorithm itself is based on state-of-the-art graph metrics. We call the proposed algorithm by the adaptive NN-based list DAG scheduling algorithm.

The remainder of the paper is organized as follows. In the Section 2 the description is presented with the potential real-world applications where DAG scheduling algorithms are applied. Section 3 describes the model of heterogeneous systems, mathematical problem statement and the process of generating random DAGs for our simulation. Section 4 describes the proposed NN-based scheduling algorithm, algorithm architecture, learning process and description of algorithms considered in the paper. Section 5 evaluates the NN-based scheduling algorithm in comparison with state-of-the-art scheduling algorithms (DONF, CPOP, HCPT, HPS, PETS) for a different workspaces (executors and DAG workflows) for small-scale and large-scale DGA, and comparison with the exact MILP solution for small-scale DAG obtained with the Guroby solver. Section 6 contains the conclusion.

2 Real-World Applications

Most real-world applications, including high performance computing applications[43], operating systems [44], machine learning applications [46], use the static DAG workflow model in which nodes represent application tasks and edges represent inter-task data dependencies.

Main components of the various real-world applications of computational problems, which can be represented as a DAG:

1. **Data model**
   - Distributed Cloud Computing and Edge computing. DAG scheduling algorithm for distributed cloud computing and edge computing should construct an optimal schedule for each new list of data input for cloud computing and edge computing. Therefore, the DAG scheduling algorithm should run again for each new data input.
   - Tensorflow. Data input for neural networks can change, but DAG scheduling algorithm only needs to be performed one time for each neural network computational graph. Therefore, the work of the algorithm does not depend on the data input.
2. **Computational model**
   - Distributed Cloud Computing and Edge computing. List of tasks coming to the distributed computing system and edge computing system should be represented as a DAG, which describes the jobs as a nodes and edges as a fixed execution sequence. Another important thing is that in the cloud computing and edge computing there could be tasks which should be implemented on a specific computational

resource (such as CPU, GPU), therefore in DAG there are several types of nodes (section 5.1).
- Tensorflow. Computational graph for neural networks is represented as a DAG. Since a computational graph for neural network does not change in time and it is known both for training and inference, then the related scheduling is a static DAG scheduling problem. Another important thing is that in the computational graph there could be several types of the nodes representing different requirements for the operations in computational graph (section 5.1).

3. **Hardware model.** The computational cluster usually consists of several working virtual machines where each one of them has several executors running (multiple CPUs and GPUs with different computational characteristics). Mobile edge computing system consisting of a mobile user and edge servers where server could also contain multiple CPUs and GPUs with different computational characteristics. To train a neural network, there can be a multi-core or many-core CPU, GPU or other accelerators. Examples of such systems are used to test the results of the proposed algorithm (section 5.1). In all these problems, it is also necessary to take into account data transferring delay or costs from one executors to another (section 3.1, formulas 9-10).

3 DAG Scheduling Problem Description

The problem addressed in this paper is the scheduling of a single DAG in distributed heterogeneous system. The scheduling model consists of three parts:

1. **Directed acyclic graph** $G = (J, E)$, where J is a set of nodes and E is the set of edges:
   - Edge $(v_i, v_j) \in E$ denotes the precedence constraint such that node v_j must wait until task v_i finishes its execution.
   - Cost of communication $b_{j,m}$ between nodes v_j and v_m, $(v_j, v_m) \in E$ should be taken into account if node v_j and v_m assigned to different executors, otherwise there is no cost of communication. If node v_m has several parents (v_j,..,v_k) that were performed on executors other than the executor assigned to node v_m, then the cost of communication are taken into account from each parent $b_{j,m} + \ldots + b_{k,m}$.
   - Each node v_j has a type that denotes on what type of executor should this node be executed.
   - Set of immediate predecessors of node v_j in a DAG is expressed as $pred(v_j)$. A node without any predecessor is called an entry node. Multiple entry nodes may exist in a DAG.
   - Set of immediate successors of node v_j is expressed as $succ(v_j)$. A node without any successor is called an exit node. There may be multiple exit nodes in a DAG.

2. **Distributed heterogeneous computing**, which consists of a set P, $p_i \in P$ heterogeneous executors with a fully connected topology:
   - D is the computation cost matrix, where $D^{j,i} = w_{j,i}$ is the execution time for executor $p_i \in P$ to process node $v_j \in J$.
   - Each executor p_i has a type that denotes what type of tasks can be defined for this executor. In order to assign a node $v_j \in J$ to a executor p_i, it is necessary that the type of node v_j and the type of executor p_i match.
3. **Performance criterion for scheduling.** Before presenting the final scheduling objective function, we first define the Makespan, Earliest Start Time (EST), Earliest Finish Time (EFT):
   - Makespan is the finish time of the last node in the scheduled DAG. It is defined by $makespan = max\{AFT(v_{exit})\}$ where $AFT(v_{exit})$ is the actual finish time (AFT) of exit node v_{exit}. In the case where there are multiple exit nodes, the makespan is the maximum AFT of all exit nodes.
   - $EST(v_j, p_i)$ denotes the earliest start time of node v_j on executor p_i and it is defined as $EST(v_j, p_i) = \max\{T_{Ava}(p_i), \max_{\{v_m \in pred(v_j)\}}\{AFT(v_m) + b_{m,j}\}\}$ where $T_{Ava}(p_i)$ is the earliest ready time of executor p_i.
   - $EFT(v_j, p_i)$ denotes the earliest finish time of node v_j on executor p_i and is defined as $EFT(v_j, p_i) = EST(v_j, p_i) + w_{j,i}$.

The objective function in the DAG scheduling problem is to determine the assignment policies for the node to heterogeneous executors so that the makespan is minimized.

3.1 Mathematical Problem Statement

In this section, our core formulation and customizations are described in detail. The parameters and variables used in the mathematical model are given in the Table 1.

The objective function in 1 minimizes the makespan over all DAG nodes. Makespan is always bigger than the completion time of any node, constraint 2. Constraint 3 is used to minimize the number of irrelevant binary variables, $X_{j,i}$ is only defined as binary when $B^{j,i} > 0$, i.e. when executor p_i offers the services necessary to complete node v_j. Constraint 4 specifies that each node must be assigned once to exactly one executor. 5 – 8 denote that the variable $Z_{j,k}$ through equation assumes that the sum of products between the variables $X_{j,i}$ and $X_{k,i}$. Constraints 9 – 10 state that a node cannot start until its predecessors are completed and data has been transmitted to it if the preceding jobs were executed on a different executor (if required). Collectively, the constraints 11 – 15 specify that an executor may process at most one node at a time and that if two nodes v_j and v_k are assigned to the same executor, their execution times may not overlap.

$$Minimize : C_{max} \tag{1}$$

Table 1: Parameters and variables of the mathematical model.

Designation	Description
	Input Parameters
Q	is a full node precedence graph. 1 if node $v_j \in J$ comes any time before node $v_m \in J$, 0 otherwise
B	1 if executor $p_i \in P$ needed to complete node $v_j \in J$, 0 otherwise
M	is an upper bound on the makespan set by default to a very large number
D	$D^{j,i} = w_{j,i}$ is the execution time for executor $p_i \in P$ to process node $v_j \in J$
	Decision Variables
X	is a matrix llocation of node to executor $X \in B^{n*m}$, where n is the number of vertices in a DAG, m is the number of executors. 1 if executor $p_i \in P$ is assigned to complete $v_j \in J$, 0 otherwise
S	is a vector scheduled start time of node, $S_j \in R^n$
θ	is a matrix support variable used to determine whether two nodes overlap, $\theta \in B^{n*n}$. 1 if node $v_m \in J$ is started before node $v_j \in J$, 0 otherwise
C_{max}	is the makespan of the schedule, $C_{max} \in R$
Z	is a matrix support variable used to assign communication costs, $Z \in B^{m*n}$. 1 if node $v_m \in J$ and node $v_j \in J$ assigned to the same executor,0 otherwise
$K_{i,j,m}$	is the product of binary variables $X_{j,i}$ and $X_{m,i}$. Since the product between variables violates the linearity of the model variable $K_{i,j,m}$, assumes its value through inequalities of model, $K_{i,j,m}$, $\in B^{m*n*n}$.

$$C_{max} \geq S_j + D^{j,i} \cdot X_{j,i},\ \forall v_j \in J,\ \forall p_i \in P : B^{j,i} > 0. \tag{2}$$

$$X_{j,i} = 0,\ \forall v_j \in J,\ \forall p_i \in P : B^{j,i} = 0. \tag{3}$$

$$\sum_{p_i \in P} X_{j,i} = 1, \forall v_j \in J. \tag{4}$$

$$Z_{j,k} = \sum_{p_i \in P} K_{i,j,k},\ \forall (v_j, v_k) \in E, \tag{5}$$

$$K_{i,j,k} \leq X_{j,i},\ \forall (v_j, v_k) \in E,\ p_i \in P, \tag{6}$$

$$K_{i,j,k} \leq X_{k,i},\ \forall (v_j, v_k) \in E,\ p_i \in P, \tag{7}$$

$$K_{i,j,k} \leq X_{k,i} + X_{j,i} - 1,\ \forall (v_j, v_k) \in E,\ p_i \in P. \tag{8}$$

$$S_k \geq S_j,\ \forall (v_j, v_k) \in E, \tag{9}$$

$$S_k \geq S_j + \sum_{p_i \in P} X_{j,i} \cdot D^{j,i} + \sum_{v_l \in pred(v_k)} b_{l,k} \cdot (1 - Z_{l,k}),\ \forall (v_j, v_k) \in E. \tag{10}$$

$$S_k - \sum_{p_i \in P} D^{j,i} \cdot X_{j,i} - S_j - \sum_{v_l \in pred(v_k)} b_{l,k} \cdot (1 - Z_{l,k}) \geq M \cdot \theta_{j,k}, \tag{11}$$

$$\forall v_j,\ v_k \in J,\ v_j \neq v_k,\ Q^{j,k} \neq 0 \tag{12}$$

$$S_k - \sum_{p_i \in P} D^{j,i} \cdot X_{j,i} - S_j - \sum_{v_l \in pred(v_k)} b_{l,k} \cdot (1 - Z_{l,k}) < M \cdot (1 - \theta_{j,k}), \tag{13}$$

$$\forall v_j, v_k \in J,\ v_j \neq v_k,\ Q^{j,k} \neq 0, \tag{14}$$

$$X_{j,i} + X_{k,i} + \theta_{j,k} + \theta_{k,j},\ \forall v_j,\ v_k \in J,\ p_i \in P. \tag{15}$$

3.2 Graph Generation

We use open-source project DAGGEN to generates random DAGs [50]. This tool relies on a layer-by-layer approach with six parameters: the number of vertices (n), a fat (f) and regularity (r) parameters for the layer sizes, and a density (d) and jump (j) parameters for the connectivity of the DAG, ccr (c) is the ratio of the sum of edges weights to the sum of the node weights.

The number of elements per each layer is uniformly drawn in an interval centered around an average value determined by the fat parameter and with a range determined by the regularity parameter. Lastly, edges are added between layers separated by a maximum number of layers determined by the jump parameter (edges only connect consecutive layers when this parameter is one). For each vertex, a uniform number of predecessors is added between one and a maximum value determined by the density parameter.

DAGs were generated with the following parameters:

- $n \in \{30, 60, 90, 36, 114, 576, 2400, 9600, 36864\}$,
- $f \in \{0.2, 0.5\}$,

- $r \in \{0.2, 0.8\}$,
- $j \in \{2, 4\}$,
- $c \in \{0.2, 0.8\}$.
- $d \in \{0.1, 0.4, 0.8\}$.

4 Algorithm

4.1 Algorithm Architecture

Figure 2 shows the overall system structure of NN-based approach to DAG scheduling. We assume that workflow description and generated DAG graph are provided to scheduling system.

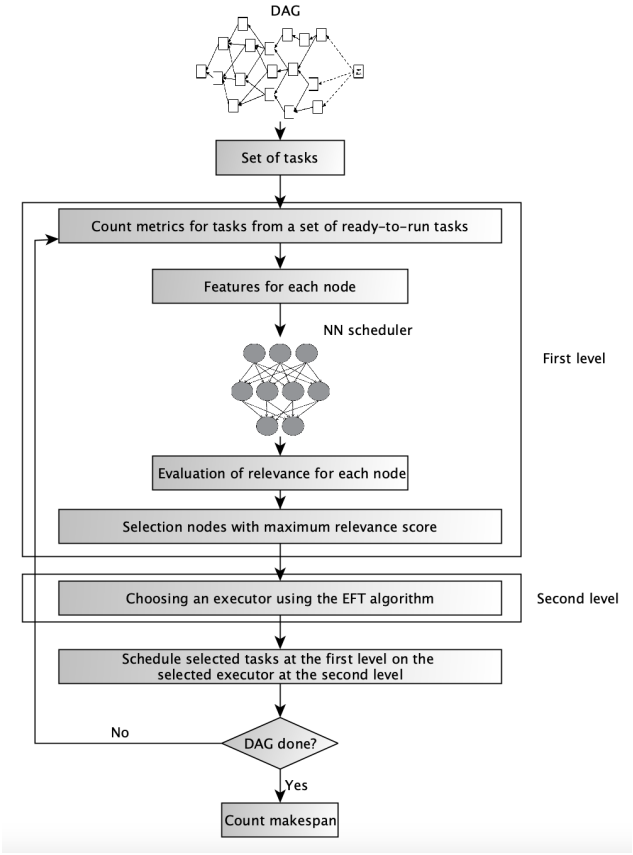


Fig. 2: Neural network for DAG scheduling

- The first level of the algorithm is the NN module based on input parameters from DAG metrics for one node from ready to perform node list. Graph scheduling metrics are described in the section 4.2. The output

of the neural network indicates the relevance of the node to be scheduled first from ready to perform node list. After receiving the relevance value for each node from ready to perform node list, the node with the maximum relevance value is selected first for execution.

- The second level of the algorithm is the assignment of a task to a computing resource (same type as the task) using EFT algorithm. The selected task at the first level is assigned to the executor defined at the second level and is passed into the set of completed tasks, then the set of tasks ready to perform node list is updated.

If all the tasks are completed, then the scheduling procedure stops. Otherwise, the algorithm repeats with an updated ready to perform node list.

4.2 Graph Scheduling Metrics

The main stage of the algorithm development is the search of metrics that could be extracted on the local scale to describe each node in the graph. A large overview of graph metrics is presented in paper [2]. Based on it, the following metrics were selected to describ the input parameters of the neural network. In the Table 2 the list of graph metrics that are used to describe each node v_i as input for the neural network.

Table 2: Graph Scheduling Metrics.

Designation	Description
$WOD(v_i)$	Weighted out-degree(WOD) of node v_i: $WOD(v_i) = \sum_{v_j \in succ(v_i)} \frac{1}{ID(v_j)}$, where $ID(v_j)$ is the in-degree of node v_j
$WOD_2(v_i)$	2-degree WOD: $WOD_2(v_i) = \sum_{v_j \in succ(v_i)} \left(\frac{1}{ID(v_j)} + \alpha \cdot \sum_{v_k \in succ(v_j)} \frac{1}{ID(v_k)} \right)$, where α is a factor of the second out-degree
$rank(v_i)$	prioritizing tasks: $rank(v_i) = \overline{w_i} + \max_{v_j \in succ(v_i)}(\overline{b_{i,j}} + rank(v_j))$
$C(v_i)$	Computational complexity of the node v_i
$\|pred(v_i)\|$	Quantity of the predecessors node v_i
$\|succ(v_i)\|$	Quantity of the successors node v_i
TW_{in}	Total weight of incoming edges in node v_i
TW_{ou}	Total weight of outgoing edges in node v_i

4.3 Training Neural Networks using Genetic Algorithms

Genetic algorithms are used for optimization and learning based on certain features of biological evolution:

1. Approach for encoding solutions of the optimization problem using chromosomes.
2. Evaluation function that returns a rating tor each chromosome given to it.
3. Approach for initializing the population of chromosomes.
4. Operators that may be applied to parents when they reproduce to alter their genetic composition. Included might be mutation, crossover (i.e. recombination of genetic material), and domain-specific operators.
5. Parameter settings for the algorithm, the operators, and so forth.

Given these five components, a genetic algorithm operates according to the following steps. Let S be the set of all possible permissible encodings. In the classical genetic algorithm, all $c \in S$ must have the same constant length Ln. A real fitness-function f defined on S is a function of solution assessment. We are searching for its point of maximum or minimum. The convergence of the genetic algorithm is based not on the analysis of the fitness-function, but on working with some set of solutions, which is called a population. Elements of the population are called individuals. Before the start of the algorithm we generate the initial population $P^0 \subset S$, which consist of m encodings $c_1^0, ..., c_m^0$. A genetic algorithm consists of iterative implementations of operations (i is the number of the generation). A detailed description of the genetic algorithm is given in Algorithm 1.

Algorithm 1 Genetic algorithm

1. Assessment: calculate $f(c)$ for all $c \in P^i$
2. Selection: choose a reproductive set $R^i \subset P^i$ depending the on previous assessment
3. Reproduction:
   (a) Crossover: choose pairs from the reproductive set R^i and make new individuals from them with some special algorithm. The resulting descendants form the set N^i
   (b) Mutation: with some probability a number of individuals $c_k \in P^i \cup N^i$ can mutate. This means they get some independence from fitness-function changes in their encoding: $c_k \longrightarrow c'_k$(they must still be permissible $\Leftrightarrow c'_k \in S$)
4. Updating the population: create a new population P^{i+1} from m individuals (usually the best) $c \in P^i \cup N^i$

4.4 Computational Complexity

Below consider a brief description of heuristics based list scheduling algorithms with their time complexity and priority attributes 3. Time complexity

is defined by the parameters v, and p where v denotes the tasks and where p is the count of processors, o is the maximum out-degree of any DAG node.

Table 3: List Scheduling Algorithms for Heterogeneous System

Algorithms	**Complexity**	**Reference to the source**
CPOP	O(v^2p)	[11]
HCPT	O(v^2p)	[28]
HPS	O($v^2(plogv)$)	[13]
PETS	O($v^2(plogv)$)	[15]
DONF	O(v×o×p)	[17]
NN	O(k×o×p)	this paper

The time complexity of the algorithm that we propose for DAG scheduling is $O(kop)$, where o is the maximum out-degree of any node, p is the number of processors and k is number of nodes in ready to perform node list. The presented algorithm is either comparable in complexity to other heuristic algorithms, for example DONF, or exceeds them. In the worst case, the proposed algorithm has complexity $O(v^2p)$, where o large as $v1$, like DONF which in worst case same complexity $O(v^2p)$. In other cases, the computational complexity of the DONF and other algorithms is larger.

4.5 State-of-the-Art DAG Scheduling Algorithms

In this section consider a detailed description of state-of-the-art DAG scheduling algorithms that are used to compare with the proposed NN-based algorithm. These algorithms are selected in accordance with the review paper for graph scheduling in heterogeneous environments [28].

Critical Path on a Processor (CPOP) algorithm [11]. Here in the first phase, the priority of a node is defined by the upward rank ($rank_u$)

$$rank_u(v_i) = \overline{w_i} + \max_{v_j \in succ(v_i)} (\overline{b_{i,j}} + rank_u(v_j)) \tag{16}$$

and downward rank ($rank_d$)

$$rank_d(v_i) = \max_{v_j \in pred(v_i)} (\overline{b_{i,j}} + \overline{w_j} + rank_d(v_j)). \tag{17}$$

The final priority of node v_i is calculated by:

$$priority(v_i) = rank_d(v_i) + rank_u(v_i). \tag{18}$$

Then the critical-path processor is defined as the processor that minimizes the cumulative computation costs of the nodes on the critical path (i.e., the longest path from any entry node to any exit node). In general, it is the one that runs

the fastest. In the processor selection phase, all nodes on the critical path are assigned to the critical-path processor. All the other nodes are assigned to the processor with minimum EFT. The computational complexity of the CPOP algorithm is $O(v^2p)$.

Heterogeneous Critical Parent Trees (HCPT) algorithm [28] is a straightforward procedure for BNP (Bounded number of heterogeneous processor) scheduling. It uses the other method in which a graph is divided into two group of unordered parent tree for scheduling that is a critical path (CN). The zero variance among average latest start time and average earliest start time is specified as a critical node. In the literature, analysis and experiments have been expressed that HCPT algorithm produced relatively better results jointly with less complexity. The computational complexity of the HCPT algorithm $O(v^2p)$.

High-Performance Task Scheduling (HPS) algorithm. The HPS algorithm [13] has 3 execution stages. To sort the nodes, the traversing of DAG is being done in top down manner. In node prioritizing phase, computation of node priority is done by using the Up Link Cost, Down Link Cost, and Link Cost node priority attributes. In processor selection phase, the minimum EFT (given by the processor) is chosen for execution of that node. The HPS algorithm uses insertion policy. The computational complexity of the HPS algorithm $O(v^2(plogv))$.

Performance Effective Task Scheduling (PETS) algorithm [15] completes its execution in three stages like HPS algorithm. In the first stage, the node sorting is done at each step of the node. The second stage is the node prioritizing stage, here node prioritization is performed using the Data Transfer Cost (DTC), Average Computation Cost (ACC), and Rank of Predecessor Task (RPT) priority attributes. The DTC is the communication cost of data transfer from the node v_i to all its instant successor tasks; for an exit node DTC $v_{exit} = 0$. The RPT denotes the instant predecessor's highest rank value of node v_i. For an entry node node RPT $v_{entry} = 0$. The rank evaluation of node v_i derives from DTC, ACC, and RPT values and represented by:

$$rank(v_i) = rank\{ACC(v_i) + DTC(v_i) + RPT(v_i)\}. \tag{19}$$

The highest rank value node is assigned highest priority. In the processor selection phase, the processor is selected using EFT approach. The PETS algorithm uses insertion policy. The computational complexity of the PETS algorithm $O(v^2(plogv))$.

Degree of Node First (DONF) algorithm [17] is feasible to maintain higher parallelism during the scheduling process in order to make full use of heterogeneous system resources. Thus the chosen node should have the property of enlarging parallelism as much as possible. Degree-of-node scheduling procedure can be shortly described as nodes with larger out-degree should be

scheduled earlier. The weighted out-degree (WOD) of node v_i is defined by:

$$WOD(v_i) = \sum_{v_j \in succ(v_i)} \frac{1}{ID(v_j)}, \tag{20}$$

where $ID(v_j)$ is the in-degree of node v_j. Next, a processor with minimizes EFT is selected. The computational complexity of the DONF algorithm $O(vop)$.

5 Numerical Simulation

In this section, the design of an experiment is explained and the performance of the proposed NN-based algorithm is evaluated. The hardware that was used in experiment is a PC with Intel Core i5-8600 with 6 cores and 3.10Ghz base frequency, 16gb of RAM and NVIDIA GeForce GTX 1060 with 6gb of vRAM.

5.1 Simulation Environment

The proposed NN-based algorithm is tested on a small-scale and large-scale DAGs to measure its generalization and robustness depending on different system configurations. Additionally for the small-scale DAGs the closeness of the makespan for state-of-the-art DAG scheduling algorithms and the global optimal makespan is analyzed. For all test cases in the simulation, data transmission starts only when all the former ones finish. The communication speed between execution nodes is 1085 MB/s.

5.1.1 Workspaces for Small-Scale DAG

The working environment for small-scale DAG is determined by different heterogeneous configurations where from three to nine executors are used. The details of the configurations are shown in Table 4.

These executors differ in computational capabilities and types:

1. 3 executors of various types are defined which differ in computational capabilities: 26, 134, and 34 GFlops respectively for the first, second and third types.
2. 6 executors, previous configuration has been expanded by the three additional performers which differ in computational capabilities: 50, 70, and 20 GFlops respectively for the first, second and third types.
3. 9 executors, here as well the second configuration is expanded by computing capacities: 125, 40, 60 GFlops respectively for the first, second and third types.

Table 4: Description workspace for small-scale DAG (30,60,90 nodes)

Executors	Type	Computational capabilities (GFlops)
	Workspace 1: 3 executor	
1	1	26
2	2	134
3	3	34
	Workspace 2: 6 executor	
1	1	26
2	2	134
3	3	34
4	1	50
5	2	70
6	3	20
	Workspace 3: 9 executor	
1	1	26
2	2	134
3	3	34
4	1	50
5	2	70
6	3	20
7	1	125
8	2	40
9	3	60

5.1.2 Workspaces for Large-Scale DAG

The working environment for large-scale DAG is determined by different heterogeneous configurations where from three to ninety six executors are used, 6 configurations in total. The types of executors are evenly distributed (equally for executors of types 1, 2, and 3), and their powers were set randomly from a range of 10 to 300 GFlops. Each workspace is intended for testing on a fixed DAG dimension. The details of the configurations test cases are shown in Table 5. A total of 6 test cases were identified for large-scale DAGs:

1. **3 executors**, 1 executor of each type (1 for 1 type, 1 for 2 type, 1 for 2 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 36 nodes.
2. **6 executors**, 2 executor of each type (2 for 1 type, 2 for 2 type, 2 for 2 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 144 nodes.
3. **12 executors**, 4 executor of each type (4 for 1 type, 4 for 2 type, 4 for 2 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 576 nodes.
4. **24 executors**, 8 executor of each type (8 for 1 type, 8 for 2 type, 8 for 2 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 2400 nodes.

5. **48 executors**, 16 executor of each type (16 for 1 type, 16 for 2 type, 16 for 2 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 9600 nodes.
6. **96 executors**, 32 executor of each type (32 for 1 type, 32 for 2 type, 32 for 2 type) which differ in computational capabilities: randomly generated from 10 to 300 GFlops. Testing is performed on DAGs of dimension 36864 nodes.

Table 5: Description test cases for large-scale DAG (36, 114, 576, 2400, 9600, 36864 nodes)

Total number executors	Number executors each type	Dimension DAGs
	Workspace 4: 3 executor	
3	1	36
	Workspace 5: 6 executor	
6	3	144
	Workspace 6: 12 executor	
12	4	576
	Workspace 7: 24 executor	
24	8	2400
	Workspace 8: 48 executor	
48	16	9600
	Workspace 9: 96 executor	
96	32	36864

5.2 Training and Testing Graph Description

For training and evaluation of the proposed NN-based algorithm batch of random DAG are used (described in Section 3.2).

At fixed dimensions exist 48 topologies. For small-scale DAGs, 3 graphs are generated for each topology for training (totally 144 for each dimension 30, 60, 90) and 10 graphs for evaluation (totally 480 for each dimension 30, 60, 90). Total number of small-scale DAGs tested was 1440. Comparative analysis is presented in the Section 5.4. Examples of small-scale DAGs are shown in Fig.3.

As in the case of small-scale DAGs, at fixed dimensions exist 48 topologies. For large-scale DAG 3000 graphs are generated for each topology for training (totally 144 000 for each dimension 36, 114, 576, 2400, 9600, 36864) and 10000 graphs for evaluation (totally 48 000 for each dimension 36, 114, 576, 2400, 9600, 36864). Total of 288 000 large-scale DAGs were tested. Examples of large-scale DAG are shown in Fig.4. Comparative analysis is presented in the Section 5.4.

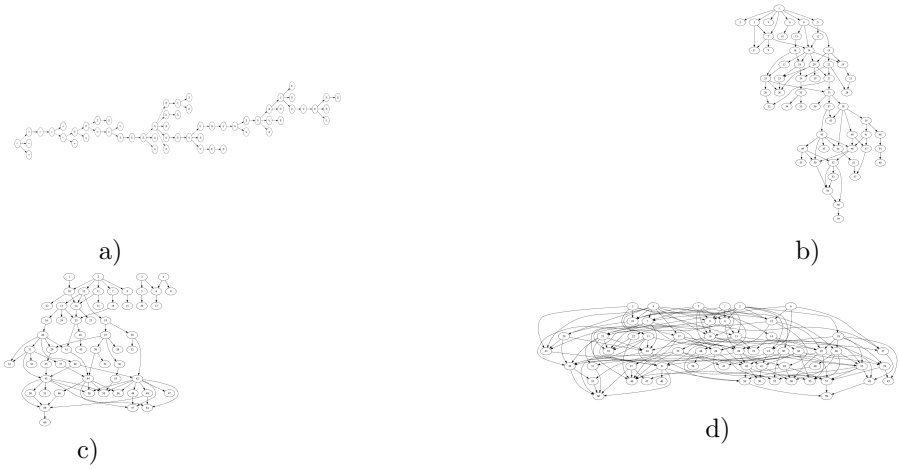


Fig. 3: Examples of DAGs with 60 vertices: a) f = 0.2, d = 0.1, r = 0.2, j = 2, c = 0.2, b) f = 0.2, d = 0.8, r = 0.2, j = 4, c = 0.8, c) f = 0.5, d = 0.4, r = 0.2, j = 4, c = 0.8, d) f = 0.5, d = 0.8, r = 0.8, j = 4, c = 0.8.

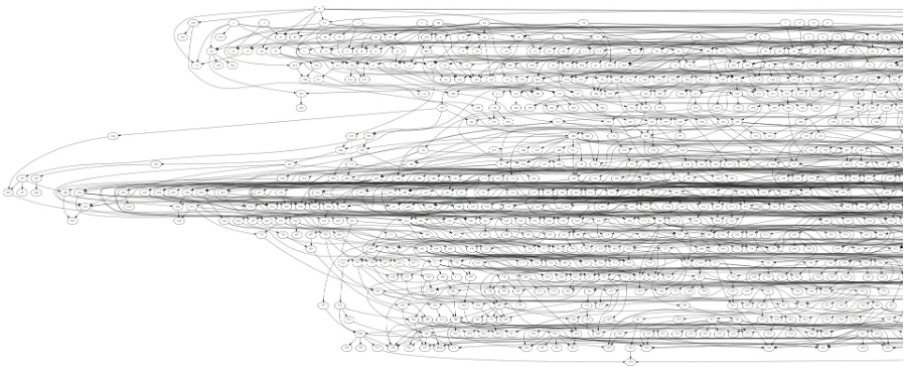


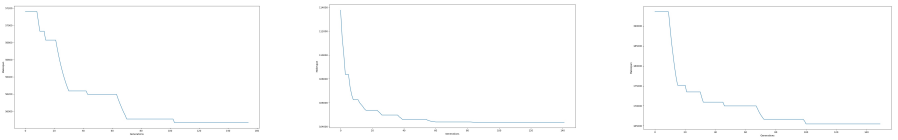
Fig. 4: Examples of large-scale DAG

5.3 Training of Neural Network

In order to perform a neural network weight optimization we use the genetic algorithm. We start by describing the five components of the algorithm listed in Section 4.3:

1. Chromosome Encoding: weights and biases in the neural network are encoded as a list of real numbers.
2. Evaluation Function: total makespan for all training graphs.
3. Initialization Procedure: weights of the initial members of the population are chosen at random with a uniform distribution between -1, 1.

4. Operators: TPI and SBX crossovers were used which were randomly selected for each individual. Mutation operator takes one parent and randomly changes some of the entries in its chromosome to create a child. Crossover operator takes two parents and creates one or two children containing some of the genetic material of each parent.
5. Parameter Settings. There are a number of parameters whose values can greatly influence the performance of the algorithm: probability of mutation is 80 %, population size is 150, number of generations is 1000, total number of weights is 95.



30 nodes DAGs 60 nodes DAGs 90 nodes DAGs

Fig. 5: Learning curve neural network with the GA learning algorithm

The training process of the NN-based algorithm is visualized in Fig.5. The generations of the genetic algorithm are located on the X-Axis, the average makespan for all graphs is located on the Y-Axis.

5.4 Result Compared with SoTA

In this subsection, the comparison results between NN-based algorithm and state-of-the-art DAG scheduling algorithms is provided: DONF, CPOP, HCPT, HPS, PETS and a comparison with the global optimal solution for each workspace on a small-scale DAGs. This is followed by a comparison of NN-based with the better scheduling algorithm on large dimension DAGs.

The comparison was carried out in accordance with the following metric for each tested graph:

$$p = \frac{(makespan_{sota} - makespan_{NN}) \cdot 100\%}{makespan_{sota}}, \tag{21}$$

where $makespan_{sota}$ is the finish time of the last node in the scheduled DAG using one of the algorithms from the list above, $makespan_{NN}$ is the finish time of the last node in the scheduled DAG using the NN-based algorithm. Obtained values are summarized and averaged for each workspace and DAG dimension.

5.4.1 Result for Small-Scale DAGs

The proposed NN-based algorithm is tested on a small-scale DAGs to measure its generalization and robustness:

- depending on different system configurations (workspace 1: 3 executors; workspace 2: 6 executors; workspace 2: 9 executors) and different DAG dimensions (30, 60, 90 nodes in DAG),
- analyzing the closeness of the makespan for a provided schedule to the global optimal makespan (defined using the MILP solution, more details in the Sections 3.1, 5.5).

Comparison results are presented in the Table 6 for each workspace. According to simulation results, the most effective from all state-of-the-art DAG scheduling algorithms is the DONF algorithm. For a small dimension of 30 nodes in workspace 1 the NN-based algorithm provides an average improvement of 4.81% by comparison DONF, while compared to other algorithms NN-based provides an average improvement of 17.5%. However, in workspaces 2 and 3 we can see that our NN-base approach is competitive with DONF, but it does not provide significant improvements compared to DONF, at the same time outperforms consistently the other baselines.

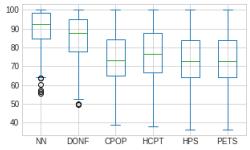
Table 6: Average percentage of improvement NN-based algorithm compared to SoTA.

Dimension	DONF	CPOP	HCPT	HPS	PETS
		Workspace 1: 3 executors			
30	**4.81**	17.75	15.04	18.31	18.41
60	**9.71**	24.14	22.01	25.11	25.03
90	**12.14**	26.07	24.86	27.59	27.6
		Workspace 2: 6 executors			
30	**2.20**	14.17	10.43	13.37	13.47
60	**6.36**	22.81	18.59	22.51	22.41
90	**9.44**	25.87	21.81	26.59	26.40
		Workspace 3: 9 executors			
30	**1.57**	10.72	6.33	8.06	8.19
60	**5.10**	20.0	14.46	17.67	17.83
90	**5.73**	21.51	16.55	20.14	20.31

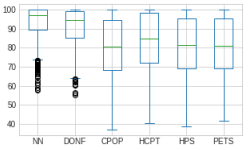
One should keep in mind that DONF is a modern heuristic, and at DAG dimensions 90 and 60 is showed the closest result to the NN-based algorithm. However the proposed algorithm shows better performance than the DONF at any other DAG dimensions. The biggest improvement is obtained on workspace 1 (on average, at all dimensions is 7.33%), and becomes less effective when moving to workspace 2 (on average, at all dimensions is 6.01%) and 3 (on average, at all dimensions is 4.13%). Also the simulation results show a tendency for the percentage improvement on NN-based algorithms over state-of-the-art algorithms to increase as the DAG dimension increases: average for all workspaces is 2.85% at dimension 30; 7.10% at dimension 60; 9.23% at dimension 90.

Fig. 6 shows a comparison of the proximity of different approaches for DAG scheduling to a global optimal solution obtained using the MILP approach. Proximity shows the percentage of how close the makespan obtained by the

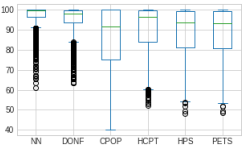
DAG scheduling algorithm to the global optimal solution obtained by the MILP approach. The implementation of the MILP solution is described in the section 5.5. According to Box Plot illustration, the median line of box for NN-based algorithm is higher than all the other algorithms, which indicates that, on average, the proposed algorithm is closest to the global optimal solution. Additionally it is important to notice that the lower performance bound for the NN-based algorithm is higher for all the other algorithms, which allows us to conclude that even in the worst conditions (low probability conditions), the NN algorithm works better. Also, the results of the algorithm are more stable, because the length of the confidence interval is the smallest and there are fewer outliers (observations that lies an abnormal distance from other values).



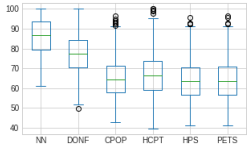
workspace 1, 30 nodes



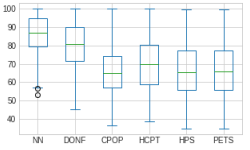
workspace 2, 30 nodes



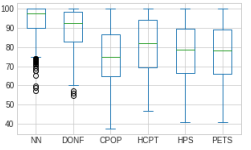
workspace 3, 30 nodes



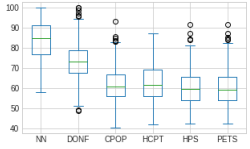
workspace 1, 60 nodes



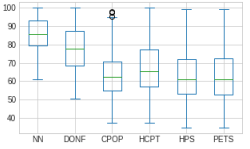
workspace 2, 60 nodes



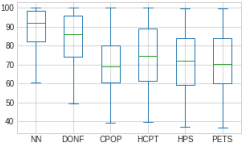
workspace 3, 60 nodes



workspace 1, 90 nodes



workspace 2, 90 nodes



workspace 3, 90 nodes

Fig. 6: Box plot with the comparison of proximity to the MILP solution for state-of-the-art DAG scheduling algorithms for each workspace and each DAG dimension.

5.4.2 Result for Large-Scale DAGs

The proposed NN-based algorithm is tested on a large-scale DAGs and is compared to the DONF algorithm which outperformed other state-of-the-art algorithms when comparing on small-scale DAGs. To determine its generalization and robustness depending on DAG dimensions and different system configurations the following list of test cases is used:

1. Test case 1: workspace 4(3 executors), DAGs of dimension 36 nodes.

2. Test case 2: workspace 5 (6 executors), DAGs of dimension 144 nodes.
3. Test case 3: workspace 6 (12 executors), DAGs of dimension 576 nodes.
4. Test case 4: workspace 7 (24 executors), DAGs of dimension 2400 nodes.
5. Test case 5: workspace 8 (48 executors), DAGs of dimension 9600 nodes.
6. Test case 6: workspace 9 (96 executors), DAGs of dimension 36864 nodes.

Fig. 7 shows a comparison of the NN-based approach compared to DONF for various test cases, where X-Axis represents the test cases and Y-Axis represents the percentage of how much the NN-based exceeds/inferior compared with DONF. On average, the proposed approach outperforms DONF by 22.77% on all test cases.

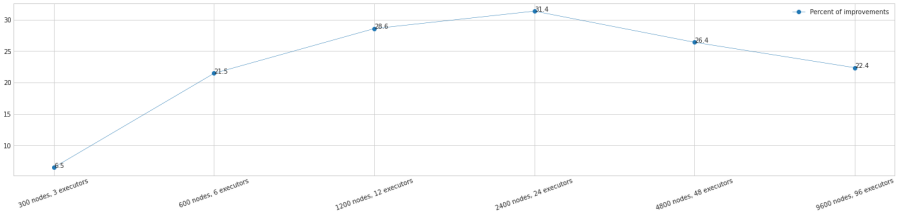


Fig. 7: Percentage improvements NN-based approach compared to DONF

The proposed NN-based algorithm is scalable and adaptive, which follows from the test results. On small-scale graphs (Test case 1, with 36 nodes and 3 executors) there is a smaller space for optimization, so the percentage of improvement was 6.51%, but as the number of nodes and executors increases, the NN-based approach increases the percentage of improvement over DONF up to 31.4%. A small decrease in performance (up to 26.43% and 24.47%, compared to DONF) on test cases 5 and 6 may be due to the need for additional model training.

5.5 Detailed Comparison

In this section a details comparison of the proposed NN-based algorithm and the best state-of-the-art algorithm (DONF algorithm) is presented. The comparison is based on the proximity of the considered algorithms to the global optimal solution calculated by the MILP approach.

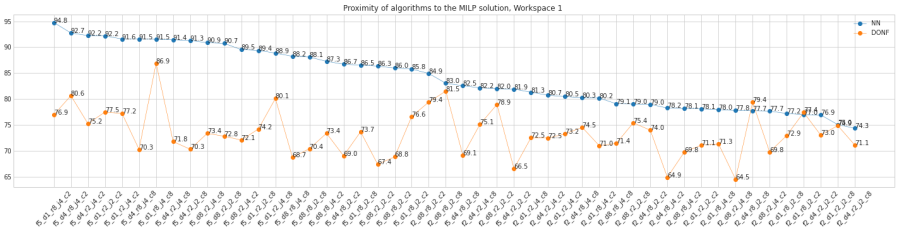
The Mixed Integer-Linear Programming formulation presented in section 3.1 is modeled and solved using Python 3.6 and MILP solver Gurobi 9.1.2 [25]. Time limit of 15 minutes, relative MILP optimality gap of 0.03, and deterministic concurrent method are used each MILP solution.

Table 7 shows the proximity results for NN-based and DONF algorithms to the MILP global optimal solution. For all the experiments NN-based algorithm provides higher proximity to the MILP solution. It is important to notice that with the help of the proposed NN-based algorithm it was possible to cover 39.2% of the proximity interval from the best DAG scheduling algorithm

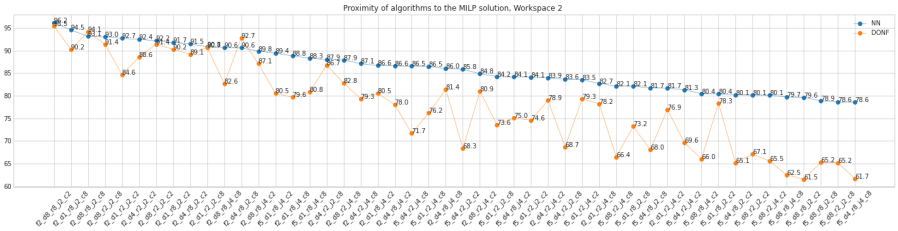
(DONF) to the global optimal solution obtained using the MILP approach: average proximity of best state-of-the-art algorithm DONF is 73.37% ; NN-based algorithm proximity is 84.08%.

Table 7: Proximity of DONF and NN-based solutions to MILP solution.

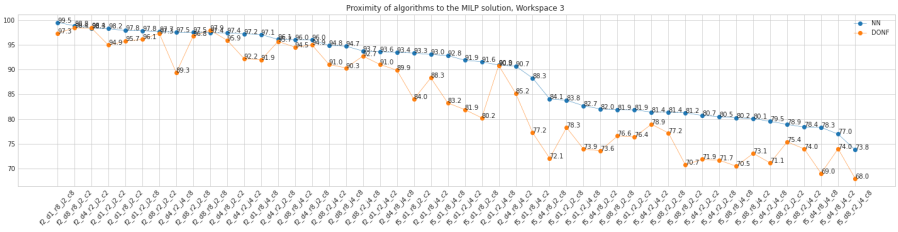
	Workspace 3		Workspace 2		Workspace 1	
Dimension	DONF	NN	DONF	NN	DONF	NN
30	94.89	**96.43**	91.13	**93.23**	85.85	**90.33**
60	89.27	**94.06**	80.66	**86.22**	77.48	**86.20**
90	84.36	**89.38**	77.82	**85.89**	73.37	**84.08**



workspace 1: 3 executors



workspace 2: 6 executors



workspace 3: 9 executors

Fig. 8: Proximity of NN-based and DONF solutions to MILP solution for DAG topology: 90.

Figure 8 shows the average proximity of NN-based and DONF solutions for each of the topologies under consideration. DAG topologies are located along the X-Axis, where the first letter denotes the generation parameter: fat(f), regularity(r), density(d), jump (j), ccr(c). The second number is the value of this parameter. For more information about the parameters and value of this parameter, see the Section 3.2.

Proximity percentages for NN-based and DONF solutions to the global optimal solution based on MILP are distributed along the Y-Axis. For workspace 1, NN-based algorithm provides an improvement in the result for all DAG topologies. In other cases, one an observe some topologies where NN-based is quite close to the DONF algorithm. It is reasonable to interpret these results as features of training on a heterogeneous sample of data. Thus, having a similar computational complexity, the NN-based algorithm provides greater efficiency in almost all the DAG topologies.

6 Conclusion

In this paper, the scheduling scheme based on artificial neural networks is investigated. This approach can be used to solve DAG static scheduling problems in heterogeneous environment where it is necessary to speed up the scheduling or to minimize the scheduling makespan.

The novelty of our work is supported by the proposed algorithm architecture based on the concept of an adaptive DAG scheduling metric, where state-of-the-art graph scheduling metrics are aggregated by the neural network. During training and selection of weights, the neural network actually becomes a new metric that adapts to the topology of the graph, which allows for more efficient scheduling for DAG in terms of reducing makespan. The first level of the algorithm is the neural network which, according to the characteristics of the nodes, determines the execution queue of the nodes. Here the set of graph metrics are used to describe the state space, allowing to extract on the local scale a description of the state of each node in the DAG. The second level of the algorithm is the assignment of a task to a computing resource of the same type as the task in accordance with the EFT algorithm. A genetic algorithm is used to train a neural network.

Simulation results show that the proposed NN-based algorithm can obtain better scheduling results than state-of-the-art heuristic rules in all static instances, the average percentage improvement on all test cases in comparison to the best algorithm DONF is 6.7% on small-scale DAGs and 22.77% on large-scale DAGs. NN-based algorithm exceed CPOP, HCPT, HPS, PETS algorithms by an average of 22.1% on small-scale DAGs. Also a comparison is presented with the globally optimal MILP solution on small-scale DAGs, where NN-based algorithm provides proximity to the global optimal solution from 84.08% to 96.43%. On large-dimensional graphs obtaining global optimal solution is possible, but the complexity grows more than quadratic, which greatly limits the scope of this approach. According to the test results the

NN-based algorithm covers 39.2% of the proximity interval from the best scheduling algorithm DONF to the global optimal solution obtained using the MILP approach.

As for future work, it is planned to extend the support of multi-dimensional executors requirements of DAGs. Furthermore, it is interesting to study the DAG scheduling problem in the online streaming settings where DAG nodes arrive dynamically over time.

References

[1] Hernandez,J. M. and Mieghem, P.V. (2011), 'Classification of graph metrics', Delft University of Technology, Tech. Rep., pp. 1–20

[2] Flint, C. and Bramas, B. (2020) 'Finding new heuristics for automated task prioritizing in heterogeneous computing'.

[3] Topcuoglu, H., Hariri, S. and Min-You Wu (2002) 'Performance-effective and low-complexity task scheduling for heterogeneous computing', IEEE Transactions on Parallel and Distributed Systems, Parallel and Distributed Systems, IEEE Transactions on, IEEE Trans. Parallel Distrib. Syst, 13(3), pp. 260–274.

[4] Ullman, J. D. (1975) 'NP-complete scheduling problems', Journal of Computer and System Sciences. Edited by C. Benzaken, 10, p. 384.

[5] Lageweg, B. J. et al. (1982) 'Computer-aided complexity classification of combinatorial problems', Communications of the ACM, 25(11), pp. 817–822.

[6] Vickson, R. G. (1984) 'Computers and Intractability: A Guide to the Theory of NP Completeness', INFOR, 22(4), p. 346.

[7] Clausen, J. (1999) 'Branch and bound algorithms-principles and examples'. Department of Computer Science, University of Copenhagen, pp. 1–30.

[8] Kohler, W. H. and Steiglitz, K. (1974) 'Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems', Journal of the ACM (JACM), 21(1), pp. 140–156.

[9] Hwang, J. J., Chow, Y. C., Anger, F. D., and Lee, C. Y. (1989) 'Scheduling precedence graphs in systems with interprocessor communication times', SIAM Journal on Computing, 18(2), pp. 244-257.

[10] Sih, G. C., and Lee, E. A. (1993) 'A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures', IEEE transactions on Parallel and Distributed systems, 4(2), pp.175-187.

[11] Arabnejad, H. (2013) 'List based task scheduling algorithms on heterogeneous systems-an overview', In Doctoral Symposium in Informatics Engineering, pp. 93.

[12] El-Rewini, H., and Lewis, T. G. (1990) 'Scheduling parallel program tasks onto arbitrary target machines', Journal of parallel and Distributed Computing, 9(2), pp.138-153.

[13] Ilavarasan, E., Thambidurai, P., and Mahilmannan, R. (2005) 'Performance effective task scheduling algorithm for heterogeneous computing system', In Parallel and Distributed Computing, The 4th International Symposium, pp. 28-38.

[14] Topcuoglu, H., Hariri, S., and Wu, M. Y. (1999) 'Task scheduling algorithms for heterogeneous processors', In Heterogeneous Computing Workshop, pp. 3-14.

[15] Ilavarasan, E., Thambidurai, P. (2007) 'Low complexity performance effective task scheduling algorithm for heterogeneous computing environments', Journal of Computer sciences, 3(2), pp. 94-103.

[16] Bittencourt, L. F., Sakellariou, R., and Madeira, E. R. (2010) 'Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm', In Parallel, Distributed and Network-Based Processing (PDP), 18th Euromicro International Conference, pp. 27-34.

[17] Lin, H., Li M., Jia C., Liu J., An H. (2019) 'Degree-of-Node Task Scheduling of Fine-Grained Parallel Programs on Heterogeneous Systems', Journal of Computer Science and Technology, 34(5), pp. 1096-1108.

[18] Sutton, R. S. (1992) 'Introduction: The Challenge of Reinforcement Learning', Machine Learning, 8(34), pp. 225–227.

[19] Heaton, J. (2018) 'Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618', Genetic Programming and Evolvable Machines, 19(1–2), pp. 305–307.

[20] Mao, H. et al. (2019) 'Learning scheduling algorithms for data processing clusters', SIGCOMM 2019 - Proceedings of the 2019 Conference of the ACM Special Interest Group on Data Communication, pp. 270–288.

[21] Sun, P.et al. (2020) 'DeepWeave: Accelerating Job Completion Time with Deep Reinforcement Learning-based Coflow Scheduling', pp. 3286–3292.

[22] Ni, X. et al. (2019) 'Generalizable Resource Allocation in Stream Processing via Deep Reinforcement Learning'.

[23] Zhang, K., Yang, Z. and Başar, T. (2019) 'Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms'.

[24] Birman, Y. et al. (2021) 'Hierarchical Deep Reinforcement Learning Approach for Multi-Objective Scheduling With Varying Queue Sizes'

[25] Gurobi Package, (2022) Gurobi Optimization. Available at: https://www.gurobi.com (Accessed: 1 March 2022).

[26] CBC Package, (2022) CBC Optimization. Available at: https://projects.coin-or.org/Cbc(Accessed: 1 March 2022).

[27] Cplex Package, (2022) Cplex Optimization. Available at: https://www.ibm.com/analytics/cplex-optimizer (Accessed: 1 March 2022).

[28] Suman, C. and Kumar, G. (2018) 'Analysis of Process Scheduling Algorithm for Multiprocessor System', 2018 7th International Conference on Reliability, pp. 564–569.

[29] Chen, S., Fang, S. and Tang, R. (2019) 'A reinforcement learning based approach for multi-projects scheduling in cloud manufacturing', International Journal of Production Research, 57(10), pp. 3080–3098.

[30] Swiechowski, M. et al. (2021) 'Monte Carlo Tree Search: A Review of Recent Modifications and Applications'.

[31] Orhean, A. I., Pop, F. and Raicu, I. (2018) 'New scheduling approach using reinforcement learning for heterogeneous distributed systems', Journal of Parallel and Distributed Computing, 117, pp. 292–302.

[32] Loth, M. et al. (2013) 'Hybridizing constraint programming and Monte-Carlo tree search: application to the job shop problem'. Edited by G. Nicosia and P. Pardalos.

[33] Jalaparti, V. et al. (2015) 'Network-Aware Scheduling for Data-Parallel Jobs: Plan When You Can', ACM SIGCOMM Computer Communication Review, 45(5), pp. 407–420.

[34] Yan, Y. et al. (no date) 'TR-Spark: Transient computing for big data analytics', Proceedings of the 7th ACM Symposium on Cloud Computing, SoCC 2016, pp. 484–496.

[35] Durillo, J. J. and Prodan, R. (2014) 'Multi-objective workflow scheduling in Amazon EC2', Cluster Computing: The Journal of Networks, Software Tools and Applications, 17(2), pp. 169–189.

[36] Soualhia, M., Khomh, F. and Tahar, S. (2015) 'Predicting Scheduling Failures in the Cloud: A Case Study with Google Clusters and Hadoop on Amazon EMR', 2015 IEEE 17th International Conference, pp. 58–65.

[37] D. G. Feitelson. (1994) 'A Survey of Scheduling in Multiprogrammed Parallel Systems', IBM TJ Watson Research Center

[38] Duan, Y., Wang, N. and W, J. (2020) 'Reducing Makespans of DAG Scheduling through Interleaving Overlapping Resource Utilization', 2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Mobile Ad Hoc and Sensor Systems (MASS), 2020 IEEE 17th International Conference on, MASS, pp. 392–400.

[39] Li, Z. et al. (2020) 'Flink-ER: An Elastic Resource-Scheduling Strategy for Processing Fluctuating Mobile Stream Data on Flink', Mobile Information Systems, pp. 1–17

[40] M. Abadi, A. Agarwal, P. Barham et al.(2015) 'TensorFlow: Large-scale machine learning on heterogeneous systems', Software available from tensorflow. org, vol. 1.

[41] Carson, J. (2017) Genetic Algorithms: Advances in Research and Applications. New York: Nova Science Publishers, Inc (Computer Science, Technology and Applications).

[42] Esfahanian, P. and Akhavan, M. (2019) 'GACNN: Training Deep Convolutional Neural Networks with Genetic Algorithm'

[43] Wang, Y. et al. (2019) 'Adaptive machine learning-based alarm reduction via edge computing for distributed intrusion detection systems', Concurrency and Computation: Practice and Experience, 31(19).

[44] Huang, B. (2019) 'Security modeling and efficient computation offloading for service workflow in mobile edge computing', Future Generation Computer System, vol. 97, pp. 755–774.

[45] Abadi, M. et al. (2016) 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems'.

[46] Apache Spark, (2022) Apache Spark. Available at: https://spark.apache.org (Accessed: 1 March 2022).

[47] Mao, Y. et al. (2017) 'A Survey on Mobile Edge Computing: The Communication Perspective'.

[48] Tang, L. et al. (2018) 'Scheduling Computation Graphs of Deep Learning Models on Manycore CPUs'.

[49] Shi, S. et al. (2018) 'A DAG Model of Synchronous Stochastic Gradient Descent in Distributed Deep Learning', 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Parallel and Distributed Systems (ICPADS), 2018 IEEE 24th International Conference on, pp. 425–432.

[50] Github, (2022) Daggen. Available at: https://github.com/frs69wq/daggen (Accessed: 1 March 2022).

Acknowledgments. This work was supported by Saint-Petersburg State University, project ID: 94062114 and a grant of state support of young Russian scientists – candidates of science (Project number MK-4674.2021.1.1.)