EN RU PROGRAM SYSTEMS: THEORY AND APPLICATIONS ISSN 2079-3316
Research Article HARDWARE AND SOFTWARE FOR DISTRIBUTED AND SUPERCOMPUTER SYSTEMS

UDC 519.687.1:004.75
10.25209/2079-3316-2026-17-3-275-314

Compact MILP Model for Scheduling Multi-Stage DAGs with Homogeneous Dependencies in the Cloud

Yury Dmitrievich **Semenov**[1]✉, Oleg Viktorovich **Sukhoroslov**[2]

[1] Higher School of Economics, Moscow, Russia
[2] Institute for Information Transmission Problems of RAS, Moscow, Russia
[1]✉ *ydsemenov@hse.ru*

Abstract. In this paper, we consider the problem of scheduling directed acyclic graphs (DAGs) of computational tasks in the cloud to minimize makespan or execution cost. An important part of research on solving challenging scheduling problems is designing mixed-integer linear programming models in order to use them with an exact solver, or as a source of linear programming relaxations. The existing MILP models for DAG scheduling in the cloud have a large number of variables and constraints, and are known to be notoriously hard to solve. While in general these models are hard to improve, DAGs encountered in practice often have remarkable structural properties that may be exploited to design better models.

We capitalize on such properties to devise a more compact and solvable MILP model for DAGs consisting of multiple stages (layers) with homogeneous data dependencies. The advantages of the proposed model are demonstrated by computational experiments. (*Linked article texts in English and in Russian*).

Key words and phrases: DAG scheduling, cloud computing, mathematical modeling, mixed-integer linear programming

2020 *Mathematics Subject Classification*: 68M20; 90C11,90C05

Acknowledgments: The research was carried out within the state assignment of Ministry of Science and Higher Education of the Russian Federation for IITP RAS.

For citation: Yury D. Semenov, Oleg V. Sukhoroslov. *Compact MILP Model for Scheduling Multi-Stage DAGs with Homogeneous Dependencies in the Cloud*. Program Systems: Theory and Applications, 2026, **17**:3(72), pp. 275–314. (*In English, in Russian*). https://psta.psiras.ru/read/psta2026_3_275-314.pdf

© Semenov Y. D., Sukhoroslov O. V. 2026

1. Introduction

Parallel and distributed applications are often modeled as directed acyclic graphs (DAGs) of computational tasks with data dependencies between them. Such applications arise in scientific computing [1], distributed machine learning [2], and big data [3]. The DAG scheduling problem involves determining for each task the machine on which it is executed and the time at which the execution starts. Traditionally, DAG scheduling algorithms were focused on minimizing the makespan, that is, the execution time of the graph. However, if the graph is executed in the cloud instead of an on-premises cluster, another important metric arises: the execution cost. While it is appealing to optimize both criteria at once, such an approach usually yields several incomparable Pareto-efficient schedules, and choosing the final schedule requires the intervention of a decision-maker, making it unsuitable for autonomous scheduling systems. Therefore we consider the single-objective case when one metric is optimized with respect to a constraint on the other metric (budget or deadline), or even without regarding the other metric whatsoever.

The main contribution of this paper is an improved MILP model for the special case of multi-stage DAGs with homogeneous data dependencies. As it will be shown later, this case is of certain practical importance. The model is compared to the general MILP model for DAG scheduling presented in [4].

2. Problem Statement

2.1. DAG Model

A directed acyclic graph $G = (T, D)$ consists of a set of tasks T (vertices of the graph) and a set of data dependencies D (edges of the graph). For each task t its execution time on each machine v is known and denoted as $p_{t,v}$. Usually, the execution time is obtained by dividing task size in flops by machine speed. However, we use a more general model because we want our model to be applicable to more complex scenarios when execution time does not scale linearly with machine performance (e.g. I/O-bound tasks).

Edges of the graph correspond to data dependencies between tasks. Each edge is represented by a tuple (i, j, w) meaning that task j requires w bytes of data produced by task i. Naturally, a task cannot start executing until it receives all required data. If the task that produced the data was executed on another machine, the data has to be sent over the network, incurring communication delay. We call two tasks independent if there is no directed path between the corresponding vertices in the DAG in any direction.

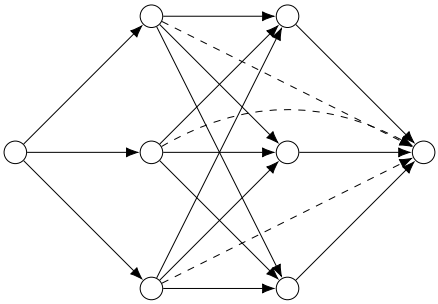


Figure 1. Example of a multi-stage DAG with homogeneous dependencies. Solid arrows denote dependencies between adjacent stages and dashed arrows denote dependencies between non-adjacent stages.

In practice, some tasks may receive external input data and may send output data to an external storage. In the existing research this is usually incorporated into the model by adding a single input task that distributes the input data and a single output task that collects the output data. We follow the same approach.

2.2. Multi-Stage DAGs with Homogeneous Dependencies

We focus our research on graphs with certain structural properties. We call a graph «multi-stage DAG with homogeneous dependencies» if the following holds:

- The vertices of the graph can be split into k stages $L_1, \ldots, L_k$ such that all tasks within one stage are independent, for all pairs of adjacent stages (L_i, L_{i+1}) any task in L_{i+1} directly depends on any task L_i, and all such dependencies have equal size.
- There may be dependencies between a non-adjacent pair of stages, but in this case the same conditions as above must apply (equal-size dependencies between all pairs of tasks).

As it will be shown later, some DAGs arising in practical applications fit this definition up to a precise approximation. By a precise approximation we mean that the dependencies may not actually have equal size, but the difference between them is negligible (e.g. tens or hundreds of bytes), not affecting network transfer time in practice.

An example of a multi-stage DAG with homogeneous dependencies is shown in Figure 1.

2.3. Cloud Resource Model

We consider the scenario when a graph is executed on a set of virtual machines (VMs) leased from an IaaS cloud provider. The most common IaaS cloud resource model used in the existing research is the following. The user has access to a set of virtual machines V where each machine belongs to one of the types offered by the provider (that is, the set of machines is usually heterogeneous, but some machines may have equivalent characteristics). Each VM type is characterized by price c_v and performance. As we have established earlier, we do not specify the performance explicitly, but define it in terms of how fast does each task execute on the given VM. We assume that a machine can execute one task at a time and the tasks cannot be interrupted during the execution.

We assume that the transfer of data dependencies between the tasks executed on different machines is performed via shared cloud storage (e.g. Amazon S3). The data produced by a task is uploaded to the storage as soon as it is computed and is fetched from the said storage on demand by dependent tasks. Transferring data of size w over the network between a machine and storage requires time w/B where B is the network bandwidth. We assume that concurrent transfers do not contend for the bandwidth. Our results can also be straightforwardly adapted to a model without shared storage.

Cloud providers usually offer VMs based on an on-demand pay-as-you-go model with the billing interval P as a parameter. If VM v is created at time svm_v and deleted at time fvm_v, the machine's lease cost is computed as

$$C_v = c_v \cdot \lceil \frac{fvm_v - svm_v}{P} \rceil. \tag{1}$$

The total cost of executing the DAG is then calculated as the sum of individual VM lease costs. We do not consider shared storage costs because for compute-intensive DAGs they are negligible compared to VM lease costs [4].

There are some natural constraints that must be satisfied when computing VM creation and deletion times. A VM must be running during all task executions scheduled on the given VM and during all data transfers to or from the given VM.

3. MILP Model

3.1. Basic Model

Before we introduce the MILP model for multi-stage DAGs with homogeneous dependencies, we describe the general MILP model for DAG scheduling in the cloud as presented in [4]. We require some additional notation.

For simplicity, we redefine dependency set D to contain tuples $(i, j, \Delta t)$ where Δt denotes data transfer time to/from storage instead of tuples (i, j, w) where w denotes data size. $\mathcal{N}$ denotes the set of all pairs (i, j) of tasks where there is no path in the DAG from i to j and vice versa.

U denotes a loose upper bound on the makespan, computed as the sum, over all tasks, of the maximum execution time of the task and the maximum data transfer time over its outgoing dependencies, including both upload to and download from the shared storage. This bound is deliberately chosen with a large safety margin, so it is sufficient for all big-M constraints used below. Finally, ε denotes a very small positive number used to convert strict inequality constraint into non-strict.

For simplicity, we assume that the graph has a single output task, denoted by *out*. If this is not the case, it is possible to introduce a new task with size 0 and add zero-weight dependencies from all output tasks to this new task.

The model contains the following decision variables:

$x_{i,v} \in \{0, 1\}$ equals 1 iff task i is placed on VM v.

$st_i \in \mathbb{R}_+$ denotes the start time of task i.

$ft_i \in \mathbb{R}_+$ denotes the finish time of task i.

$\omega_{i,j} \in \{0, 1\}$ captures overlap between independent tasks i and j. If the executions of these two tasks overlap, then both $\omega_{i,j}$ and $\omega_{j,i}$ are set to 1.

$svm_v \in \mathbb{R}_+$ denotes the start time of VM v.

$fvm_v \in \mathbb{R}_+$ denotes the finish time of VM v.

$I_v \in \mathbb{Z}_+$ denotes the number of billing intervals used by VM v.

$M \in \mathbb{R}_+$ denotes the makespan.

$C \in \mathbb{R}_+$ denotes the execution cost.

The model has the following constraints:

$$\sum_{v \in V} x_{i,v} = 1 \qquad \forall i \in T \tag{2}$$

$$st_i + \sum_{v \in V} p_{i,v} \cdot x_{i,v} = ft_i \qquad \forall i \in T \tag{3}$$

$$ft_j - st_i \leqslant U \cdot \omega_{i,j} \qquad \forall (i,j) \in \mathcal{N} \tag{4}$$

$$\omega_{i,j} + \omega_{j,i} + x_{i,v} + x_{j,v} \leqslant 3 \qquad \forall (i,j) \in \mathcal{N}\ \forall v \in V \tag{5}$$

$$svm_v \leqslant fvm_v \qquad \forall v \in V \tag{6}$$

$$svm_v \leqslant st_i + U \cdot (1 - x_{i,v}) \qquad \forall v \in V\ \forall t \in T \tag{7}$$

$$fvm_v + U \cdot (1 - x_{i,v}) \geqslant ft_i \qquad \forall v \in V\ \forall i \in T \tag{8}$$

$$I_v \cdot P \geqslant fvm_v - svm_v \qquad \forall v \in V \tag{9}$$

$$(I_v - 1) \cdot P \leqslant fvm_v - svm_v - \varepsilon \qquad \forall v \in V \tag{10}$$

$$M = ft_{out} \tag{11}$$

$$C = \sum_{v \in V} c_v \cdot I_v \tag{12}$$

$$st_s \geqslant ft_p + 2\Delta t \cdot (x_{p,v} - x_{s,v}) \qquad \forall v \in V\ \forall (p,s,\Delta t) \in D \tag{13}$$

$$svm_v \leqslant st_s + U \cdot (1 - x_{s,v}) - \Delta t \cdot (x_{s,v} - x_{p,v}) \qquad \forall v \in V\ \forall (p,s,\Delta t) \in D \tag{14}$$

$$fvm_v + U \cdot (1 - x_{p,v}) \geqslant ft_p + \Delta t \cdot (x_{p,v} - x_{s,v}) \qquad \forall v \in V\ \forall (p,s,\Delta t) \in D \tag{15}$$

Constraint 2 ensures that each task is executed on exactly one VM. Constraint 3 relates task finish times to task start times. Constraints 4 and 5 ensure that the overlap variables are correct. Constraints 6-8 set VM start and finish times, not including the data transfers. Constraints 9 and 10 ensure that the number of billing intervals is counted correctly. Constraints 11 and 12 define makespan and execution cost respectively. The constraints that account for data transfers are different for each data transfer strategy. Constraint 13 ensures that each task does not start before receiving all data from its parents. Constraint 14 ensures that all data transfers to a VM lie within its lifetime. Constraint 15 does the same, but with the transfers from a VM. Finally, the objective functions, as well as the budget and deadline constraints, are trivially expressed using the C and M variables.

Variables and constraints presented above may also be used to build a classic (makespan-only) DAG scheduling model. To do this, one should remove svm_v, fvm_v, I_v, C variables. As for constraints, 2-5, 11, and 13 should be kept.

3.2. Multi-Stage Model

In what follows, we assume that there is an arbitrary linear order on the set of VMs V. The VMs are compared according to this order and are numbered according to this order with numbers $1, \ldots, |V|$. By a slight abuse of notation, we will sometimes conflate the VMs in the set V with indices $1, \ldots, |V|$.

The key observation that allows us to build more compact models for multi-stage homogeneous DAGs is the following.

Lemma 1. *For any correct schedule, the following properties hold:*

1. *If $i < i'$, then each task $j \in L_i$ ends no later than each task $j' \in L_{i'}$ starts.*
2. *If tasks j and j' of stage L_i are executed on VM v with no other tasks in between, it is always possible and optimal to start j' as soon as j finishes.*
3. *Suppose that task j of stage L_i starts on VM v at time t. If task $j' \in L_i$ starts immediately after j, they can be interchanged so that j' starts at time t and j starts immediately after j'. This operation keeps the schedule correct and changes neither makespan nor cost.*

Proof. *1*. By definition, there are dependencies between all pairs of tasks for all pairs of adjacent stages. It implies that if $i < i'$, each task in $L_{i'}$ directly or indirectly depends on each task in L_i, therefore each task in $L_{i'}$ cannot start until all tasks in L_i are finished.

2. By definition, tasks j and j' belonging to the same stage L_i have the same set of predecessors, and for each predecessor the size of data sent to j and j' is assumed to be equal. It means that if task j received all required data by time t, then task j' can receive its data as well. Therefore, j' can start immediately after j. Removing the gap between the tasks cannot prolong the lifetime of any VM, therefore neither makespan nor cost is increased.

3. By the same logic as above, both j and j' received all required data by time t, so both of them can start at time t. As for the tasks dependent on j and j', each such task depends on both j and j' and the dependencies have equal size. Since in both cases the last of j and j' finishes at time $t + p_{j,v} + p_{j',v}$, the interchange does not affect the dependent tasks. The makespan and the cost remain the same because no VM lifetime is affected by the interchange. □

The lemma basically states that for almost all purposes we only need to consider time segments $[l_{i,v}, r_{i,v}]$ when tasks from stage i execute on VM v. Within the segment the tasks are executed without idle gaps, and the exact order of tasks does not matter at all because they can be freely interchanged. By using this idea, we base our model on the following decision variables:

$x_{j,v} \in \{0, 1\}$ equals 1 iff task j is placed on VM v.

$l_{i,v} \in \mathbb{R}_+$ denotes the start time of stage i on VM v.

$r_{i,v} \in \mathbb{R}_+$ denotes the finish time of stage i on VM v. If no task from stage i is executed on v, $r_{i,v} = l_{i,v}$.

$r^{<}_{i,v} \in \mathbb{R}$ denotes the maximum finish time of stage i on any VM $v' < v$ or some small negative number if no VM $v' < v$ executes stage i.

$r^{>}_{i,v} \in \mathbb{R}$ denotes the maximum finish time of stage i on any VM $v' > v$ or some small negative number if no VM $v' > v$ executes stage i.

$h_{i,v} \in \{0, 1\}$ equals 1 iff any task from L_i is placed on VM v.

$\widehat{h}_{i,v} \in \{0, 1\}$ equals 1 iff any task from L_i is not placed on VM v.

$svm_v \in \mathbb{R}_+$ denotes the start time of VM v.

$fvm_v \in \mathbb{R}_+$ denotes the finish time of VM v.

$I_v \in \mathbb{Z}_+$ denotes the number of billing intervals used by VM v.

$M \in \mathbb{R}_+$ denotes the makespan.

$C \in \mathbb{R}_+$ denotes the execution cost.

In what follows, given stage i by S_i we denote the set of pairs $(i', \Delta t)$ such that there are edges from all tasks in L_i to all tasks in $L_{i'}$ with transmission time Δt. Analogously, we denote the set of pairs $(i', \Delta t)$ such that there are edges from all tasks in $L_{i'}$ to all tasks in L_i with transmission time Δt by P_i. The layers are numbered from 1 to k and $[i, k]$ denotes numbers $i, i + 1, \ldots, k$. U denotes the same makespan upper bound as in the basic model. The model has the following constraints:

$$\sum_{v \in V} x_{j,v} = 1 \qquad \forall j \in T \tag{16}$$

$$l_{i,v} \geqslant r_{i-1,v} \qquad \forall i \in [2,k]\ \forall v \in V \tag{17}$$

$$l_{i,v} + \sum_{j \in L_i} p_{j,v} x_{j,v} = r_{i,v} \qquad \forall i \in [1,k]\ \forall v \in V \tag{18}$$

$$h_{i,v} \leqslant \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1,k]\ \forall v \in V \tag{19}$$

$$|L_i| \cdot h_{i,v} \geqslant \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1,k]\ \forall v \in V \tag{20}$$

$$r^{<}_{i,v} \geqslant r^{<}_{i,v-1} \qquad \forall i \in [1,k]\ \forall v \in V, v > 1 \tag{21}$$

$$r^{<}_{i,v} \geqslant r_{i,v-1} - U \cdot (1 - h_{i,v-1}) \qquad \forall i \in [1,k]\ \forall v \in V, v > 1 \tag{22}$$

$$r^{>}_{i,v} \geqslant r^{>}_{i,v+1} \qquad \forall i \in [1,k]\ \forall v \in V, v < |V| \tag{23}$$

$$r^{>}_{i,v} \geqslant r_{i,v+1} - U \cdot (1 - h_{i,v+1}) \qquad \forall i \in [1,k]\ \forall v \in V, v < |V| \tag{24}$$

$$l_{i,v} + U \cdot (1 - h_{i,v}) \geqslant r^{<}_{i',v} + 2\Delta t \qquad \forall i \in [1,k]\ \forall (i', \Delta t) \in P_i\ \forall v \in V \tag{25}$$

$$l_{i,v} + U \cdot (1 - h_{i,v}) \geqslant r^{>}_{i',v} + 2\Delta t \qquad \forall i \in [1,k]\ \forall (i', \Delta t) \in P_i\ \forall v \in V \tag{26}$$

$$\widehat{h}_{i,v} \leqslant |L_i| - \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1,k]\ \forall v \in V \tag{27}$$

$$|L_i| \cdot \widehat{h}_{i,v} \geqslant |L_i| - \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1,k]\ \forall v \in V \tag{28}$$

$$svm_v \leqslant l_{1,v} \qquad \forall v \in V \tag{29}$$

$$fvm_v \geqslant r_{k,v} \qquad \forall v \in V \tag{30}$$

$$svm_v \leqslant l_{i,v} - \Delta t \cdot (h_{i,v} + \widehat{h}_{i',v} - 1) \qquad \forall i \in [1,k]\ \forall (i', \Delta t) \in P_i\ \forall v \in V \tag{31}$$

$$fvm_v \geqslant r_{i,v} + \Delta t \cdot (h_{i,v} + \widehat{h}_{i',v} - 1) \tag{32}$$

$$\forall i \in [1,k]\ \forall (i',\Delta t) \in S_i\ \forall v \in V$$

$$I_v \cdot P \geqslant fvm_v - svm_v \qquad \forall v \in V \tag{33}$$

$$(I_v - 1) \cdot P \leqslant fvm_v - svm_v - \varepsilon \qquad \forall v \in V \tag{34}$$

$$M \geqslant r_{k,v} \qquad \forall v \in V \tag{35}$$

$$C = \sum_{v \in V} c_v \cdot I_v \tag{36}$$

Constraint (16) ensures that each task is executed on exactly one VM. Constraint (17) ensures that each subsequent stage is executed after the previous one. Constraint (18) relates start and finish times for each stage. If no tasks of stage i are executed on VM v, we have $l_{i,v} = r_{i,v}$.

Constraints (19) and (20) define the $h_{i,v}$ variables. Constraints (21)-(24) define the $r^{<}_{i,v}$ and the $r^{>}_{i,v}$ variables. They work by computing running maxima of $r_{i,v}$ for v that execute at least one task of stage i over prefixes and suffixes of V.

Constraints (25) and (26) ensure that each task receives all required data before executing. Constraints (27) and (28) define the $\widehat{h}_{i,v}$ variables.

Constraints (29) and (30) ensure that VM lifetime includes all tasks executed on that VM. Coupled with (17), they additionally ensure that $svm_v \leqslant fvm_v$.

Constraint (31) ensures that all data transfers to a VM lie within its lifetime. Constraint (32) does the same, but with the transfers from a VM.

Constraints (33) and (34) ensure that the number of billing intervals is counted correctly. Finally, constraints (35) and (36) define makespan and cost respectively.

3.3. Comparison

Consider a DAG with k stages and l pairs of stages with edges between them. In practice $k \ll |T|$ and $l \ll |D|$. The basic model has $O(|T||V| + |T|^2)$ integer variables and $O(|V||T|^2 + |V||D|)$ constraints. Note that $|T|^2$ is a bound on the size of the set $\mathcal{N}$ of pairs of independent tasks, and in general $|T|^2$ may be an overly pessimistic bound (as an extreme example, consider a chain DAG, which has $\mathcal{N} = \varnothing$). However, for DAGs encountered in practice the size of $\mathcal{N}$ is often $\Theta(|T|^2)$.

In contrast, our model has $O(|T||V|)$ integer variables and $O(|T| + (k+l)|V|)$ constraints.

Another important metric is the number of big-M constraints, since these are known to impact solver performance negatively. The basic model has $O(|T|^2 + |V|(|T| + |D|))$ big-M constraints, while the multi-stage model has only $O((k+l)|V|)$.

4. Experimental Evaluation

4.1. Dataset

A widely used dataset in modern DAG scheduling research is WfInstances [1]. WfInstances contains DAGs collected from 9 real-world distributed scientific computing applications. While the DAGs in the dataset do not precisely fit our definition of a multi-stage DAG with homogeneous dependencies, some of them fit it approximately. By approximately, we mean the following:

(1) Not all edges between a pair of stages are exactly equal, but the transmission time for different edges differs by at most 10^{-3} seconds (assuming typical cloud bandwidth of 5 Gbps).

In this case we can set all transmission times equal to their mean at the expense of a small error, which can be discarded in practice (in the worst case makespan error will be at most $10^{-3}n$ seconds). It should be noted that taking the mean transmission time may make the resulting schedule infeasible in some rare cases.

This issue can be mitigated by taking a more conservative approach and setting all transmission times to their maximum instead. However, it has negligible effect on the resulting MILP model, and in the experiments below we use mean transmission time. For applications requiring guaranteed feasibility with respect to the original DAG, the maximum transmission time can be used instead without changing the formulation.

(2) There may be a pair of stages such that some tasks are directly dependent and some are not, but in this case one of the stages is either the first stage consisting of a single input task or the last stage consisting of a single output task, and the existing dependencies have near-zero transmission time (at most 10^{-3} seconds).

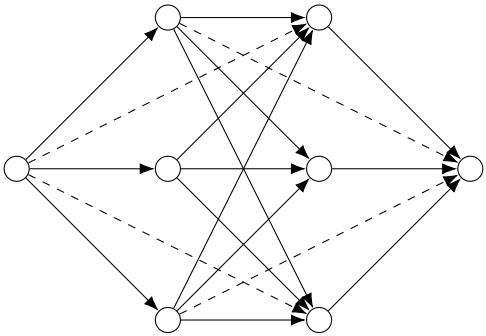


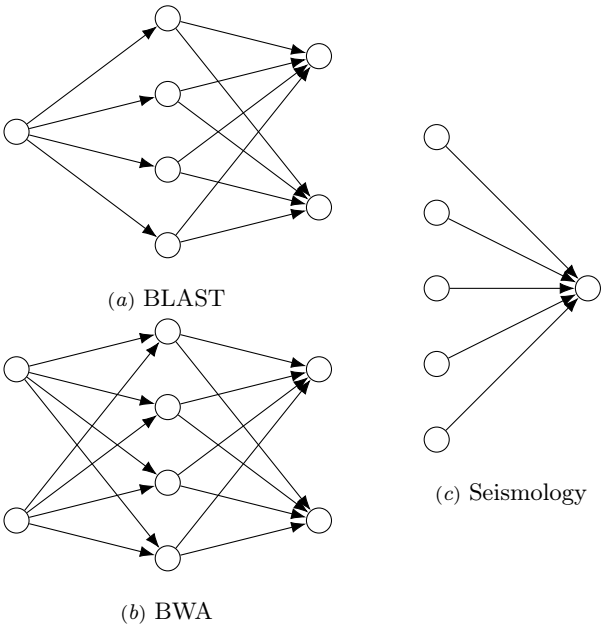
FIGURE 2. An approximately multi-stage DAG with homogeneous dependencies may not have the full set of all-to-all dependencies between the input task and some layer or between some layer and the output task, but these dependencies must have very small size.

In this case we can add the missing edges with zero transmission time. The insight behind this procedure is that the tasks that do not receive external input or send output still communicate their status to the scheduler, and it can be viewed as external input/output of negligible size. Thus, adding extra zero-size edges from the input task or to the output task does not make the model unrealistic. An example is shown in Figure 2.

Analyzing the WfInstances dataset, we found the following applications that approximately fit our special case:

- **BLAST.** DAGs belonging to this bioinformatics application have a simple three-stage fork-join structure.
- **BWA.** Similar to BLAST, but the first stage has two tasks instead of one.
- **Seismology.** These DAGs have simple two-stage structure: the first stage consists of parallel tasks, and the second stage is just a single task that processes and merges the results of the previous tasks.

Examples showing the structure of BLAST, BWA, and Seismology graphs are shown in Figure 3. For clarity, the graphs are drawn without the artificial input and output tasks.



(*a*) BLAST

(*b*) BWA

(*c*) Seismology

Figure 3. Graph structure examples

4.2. Experimental Setup

We compare the performance of two MILP models on WfInstances DAGs from the three applications mentioned above. We limit ourselves to DAGs with no more than 110 tasks, including extra input and output tasks. Following [4], we consider a set of VMs for 4 cheapest VM types in each of the Amazon EC2 m5 and c5 families, 3 VMs for each type (24 VMs in total). The billing interval is 1 second.

The models were solved with CPLEX 20.1 on Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz with 128 GB RAM. The solver was allowed to use up to 32 parallel threads. The time limit for solving the model was 1 hour.

We test three problem formulations: makespan minimization without additional constraints, deadline-constrained cost minimization, and budget-constrained makespan minimization. The last two formulations require setting a deadline or a budget.

TABLE 1. Makespan minimization model size in the number of integer variables and constraints

DAG	Model size			
	baseline MILP		multi-stage MILP	
	$I^{(1)}$	constraints	$I^{(1)}$	constraints
unconstrained				
blast-chameleon-large-*	13545	257480	3231	1573
blast-chameleon-small-*	3105	43100	1791	1513
bwa-chameleon-small-*	13780	259956	3255	1574
seismology-chameleon-100p-*	13081	252554	3041	1239
deadline/budget-constrained				
blast-chameleon-large-*	13570	277042	3256	1959
blast-chameleon-small-*	3130	51142	1816	1899
bwa-chameleon-small-*	13805	284366	3280	1960
seismology-chameleon-100p-*	13106	262372	3066	1529

(1) Number of integer variables

The standard approach [4–6] is to take some crude estimation of optimal makespan or cost and to convert it into a constraint by multiplying by some pre-defined parameter α. Cost estimation was performed by running all tasks sequentially on the cheapest VM type, as in [4, 5]. Makespan estimation was performed by running each task on a separate instance of the fastest VM type, assuming that data transfers are instant, as in [4, 6].

Parameter α was set to 10 for deadline and to 2 for budget. The reason behind taking different α for different constraints is that setting α too low would make the problem trivially infeasible and setting α too high would make the constraint trivially satisfiable and therefore useless. What is too low or too high depends on the quality of estimation, which is different for makespan and cost, therefore different parameters.

4.3. Results

Table 1 show the number of integer variables and constraints in the MILP models for unconstrained and deadline/budget-constrained problems respectively (note that both deadline-constrained cost minimization and budget-constrained makespan minimization have MILP models of equal size).

Tables 2, 3, and 4 present model solution time, solver gap, and solver status for unconstrained makespan minimization, deadline-constrained cost minimization, and budget-constrained makespan minimization respectively.

Solver gap is calculated as $\frac{best-bound}{best} \cdot 100\%$ where *best* is the best schedule found by the solver and *bound* is the best lower bound on the optimal objective value found by the solver. If the model is solved to optimality, then the gap is 0. If the model found no feasible solution, then the gap is ∞. If the gap is ∞ and the solver terminated before reaching the time limit, it means that the problem is provably infeasible.

Solver status is one of the following:

"optimal" if the model was solved to optimality,

"feasible" if some feasible solution was found, but not proved to be optimal,

"infeasible" if the solver proved that no feasible solution exists, and

"unknown" if no feasible solution was found, but infeasibility couldn't be proved either.

The solver failed to solve the basic MILP model to optimality for any problem on any DAG within 1 hour. The multi-stage model was solved on *blast-chameleon-small-** DAGs (with a few exceptions), seismology DAG (except for makespan minimization), and on two BWA DAGs in the case of deadline-constrained cost minimization.

Even though in most cases multi-stage model cannot be solved to optimality within 1 hour, for BLAST and seismology DAGs it produces solutions with an optimality gap below 4% (with a single exception of budget-constrained optimization on *blast-chameleon-small-001*) if the problem is feasible (and in many cases the gap is less than 1%). Its performance on BWA DAGs is somewhat degraded, but it still produced a smaller gap than the basic MILP model.

Basic MILP model on the other hand often produces solutions with more than 60% gap, and for deadline-constrained optimization the gap is never below 90%. As for the model size, we achieved up to 200x reduction in the number of constraints and up to 4x reduction in the number of integer variables.

TABLE 2. Makespan minimization solution time, status, and gap

DAG	baseline MILP			multi-stage MILP		
	time (s)	status	gap (%)	time (s)	status	gap (%)
blast-chameleon-large-001	3600.00	feasible	90.63	3600.00	feasible	0.25
blast-chameleon-large-002	3600.00	feasible	92.64	3600.00	feasible	0.12
blast-chameleon-large-003	3600.00	feasible	95.44	3600.00	feasible	0.24
blast-chameleon-large-004	3600.00	feasible	93.08	3600.00	feasible	0.17
blast-chameleon-large-005	3600.00	feasible	93.37	3600.00	feasible	0.17
blast-chameleon-small-001	3600.00	feasible	80.22	3600.00	feasible	3.09
blast-chameleon-small-002	3600.00	feasible	71.92	20.40	optimal	0.00
blast-chameleon-small-003	3600.00	feasible	69.96	1578.76	optimal	0.00
blast-chameleon-small-004	3600.00	feasible	71.43	638.41	optimal	0.00
blast-chameleon-small-005	3600.00	feasible	71.31	42.10	optimal	0.00
bwa-chameleon-small-001	3600.00	feasible	19.23	3600.00	feasible	4.81
bwa-chameleon-small-002	3600.00	feasible	21.95	3600.00	feasible	9.91
bwa-chameleon-small-003	3600.00	feasible	23.34	3600.00	feasible	15.50
bwa-chameleon-small-004	3600.00	feasible	21.29	3600.00	feasible	11.50
bwa-chameleon-small-005	3600.00	feasible	21.04	3600.00	feasible	7.49
seismology-chameleon-100p-001	3600.00	feasible	65.79	3600.00	feasible	0.14

Table 3. Cost minimization w.r.t. deadline ($\alpha = 10$) solution time, status, and gap

DAG	baseline MILP			multi-stage MILP		
	time (s)	status	gap (%)	time (s)	status	gap (%)
blast-chameleon-large-001	3600.00	unknown	∞	3600.00	feasible	0.15
blast-chameleon-large-002	3600.00	unknown	∞	3600.00	feasible	0.07
blast-chameleon-large-003	3600.00	unknown	∞	3600.00	feasible	0.17
blast-chameleon-large-004	3600.00	unknown	∞	3600.00	feasible	0.18
blast-chameleon-large-005	3600.00	unknown	∞	3600.00	feasible	0.11
blast-chameleon-small-001	3600.00	unknown	∞	0.07	infeasible	∞
blast-chameleon-small-002	3600.00	feasible	93.22	59.95	optimal	0.00
blast-chameleon-small-003	3600.00	feasible	93.34	3222.04	optimal	0.00
blast-chameleon-small-004	3600.00	feasible	92.86	3600.00	feasible	2.34
blast-chameleon-small-005	3600.00	feasible	93.45	3600.00	feasible	0.13
bwa-chameleon-small-001	3600.00	feasible	100.00	42.41	optimal	0.00
bwa-chameleon-small-002	3600.00	feasible	100.00	3600.00	feasible	0.01
bwa-chameleon-small-003	3600.00	feasible	100.00	3600.00	feasible	1.27
bwa-chameleon-small-004	3600.00	feasible	100.00	16.40	optimal	0.00
bwa-chameleon-small-005	3600.00	feasible	100.00	3600.00	feasible	1.34
seismology-chameleon-100p-001	3600.00	feasible	100.00	75.13	optimal	0.00

TABLE 4. Makespan minimization w.r.t. budget ($\alpha = 2$) solution time, status, and gap

DAG	baseline MILP			multi-stage MILP		
	time (s)	status	gap (%)	time (s)	status	gap (%)
blast-chameleon-large-001	3600.00	unknown	∞	3600.00	feasible	0.52
blast-chameleon-large-002	3600.00	feasible	96.74	3600.00	feasible	0.20
blast-chameleon-large-003	3600.00	feasible	95.78	3600.00	feasible	0.24
blast-chameleon-large-004	3600.00	unknown	∞	3600.00	feasible	0.37
blast-chameleon-large-005	3600.00	unknown	∞	3600.00	feasible	0.19
blast-chameleon-small-001	3600.00	feasible	76.62	3600.00	feasible	8.76
blast-chameleon-small-002	3600.00	feasible	74.32	44.74	optimal	0.00
blast-chameleon-small-003	3600.00	feasible	72.85	2194.55	optimal	0.00
blast-chameleon-small-004	3600.00	feasible	71.63	109.33	optimal	0.00
blast-chameleon-small-005	3600.00	feasible	73.26	29.13	optimal	0.00
bwa-chameleon-small-001	3600.00	feasible	22.49	3600.00	feasible	2.54
bwa-chameleon-small-002	3600.00	feasible	72.44	3600.00	feasible	22.73
bwa-chameleon-small-003	3600.00	unknown	∞	3600.00	feasible	5.85
bwa-chameleon-small-004	3600.00	unknown	∞	3600.00	feasible	11.60
bwa-chameleon-small-005	3600.00	unknown	∞	3600.00	feasible	13.19
seismology-chameleon-100p-001	3600.00	feasible	99.58	821.60	optimal	0.00

5. Conclusion

In this paper, we present a novel compact MILP model for scheduling multi-stage DAGs with homogeneous dependencies in the cloud. This model is much smaller than the conventional one due to stage compression. The experiments show the advantage of the proposed model: it significantly reduces the number of integer variables and constraints, and in most cases produces optimal or near-optimal feasible solutions within the 1-hour time limit. A promising future direction is strengthening the formulation to find optimal solutions in more cases.

References

[1] T. Coleman, H. Casanova, L. Pottier, M. Kaushik, E. Deelman, R. Ferreira da Silva. "Wfcommons: A framework for enabling scientific workflow research and development", *Future Generation Computer Systems*, **128** (March 2022), pp. 16–27. doi ↑276, 285

[2] R. Mayer, Ch. Mayer, L. Laich. The tensorflow partitioning and scheduling problem: it's the critical path! *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*, DIDL'17 (Las Vegas, Nevada, USA, December 11–15, 2017), ACM, New York, 2017, ISBN 978-1-4503-5169-0, pp. 1–6. doi ↑276

[3] M.-M. Aseman-Manzar, S. Karimian-Aliabadi, R. Entezari-Maleki, B. Egger, A. Movaghar. "Cost-aware resource recommendation for DAG-based big data workflows: An apache spark case study", *IEEE Transactions on Services Computing*, **16**:3 (2023), pp. 1726–1737. doi ↑276

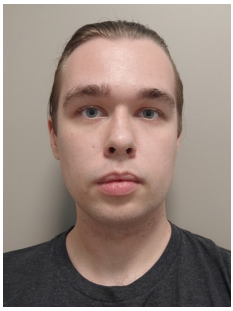
[4] Yu. Semenov, O. Sukhoroslov. "MIP models and complexity results for DAG scheduling in the cloud", *Mathematical Optimization Theory and Operations Research*, MOTOR 2025 (Novosibirsk, Russia, July 7–11, 2025), Lecture Notes in Computer Science, vol. **15681**, eds. Kochetov Yu., Khachay M., Eremeev A., Pardalos P., Springer, Cham, 2025, ISBN 978-3-031-97076-4, pp. 317–331. doi ↑276, 278, 279, 287, 288

[5] H. Li, D. Wang, nizares Abreu J. R. Ca Q. Zhao, O. Bonilla Pineda. "PSO+LOA: hybrid constrained optimization for scheduling scientific workflows in the cloud", *The Journal of Supercomputing*, **77**:11 (2021), pp. 13139–13165. doi ↑288

[6] P. Shobeiri, M. A. Rastaghi, S. Abrishami, B. Shobiri. "PCP-ACO: a hybrid deadline-constrained workflow scheduling algorithm for cloud environment", *The Journal of Supercomputing*, **80**:6 (2024), pp. 7750–7780. doi ↑288

Received 29.05.2026;
approved after reviewing 29.05.2026;
accepted for publication 28.07.2026;
published online 21.09.2026.

Information about the authors:



Yury Dmitrievich Semenov

Yury Semenov is a Ph.D. student at the Higher School of Economics. His scientific interests include distributed computing and combinatorial optimization.

0009-0004-8996-4857

e-mail: *ydsemenov@hse.ru*



Oleg Viktorovich Sukhoroslov

Ph.D., Senior Researcher at the Institute for Information Transmission Problems of the Russian Academy of Sciences, Associate Professor at HSE University. Research interests: distributed computing, cloud computing, resource management, scheduling and simulation.

0000-0003-2836-2524

e-mail: *sukhoroslov@iitp.ru*

Authors contribution: *Yury D. Semenov* — 80% (methodology, software, validation, investigation, writing (original draft)); *Oleg V. Sukhoroslov* — 20% (resources, writing (review & editing), project administration).

The authors declare no conflicts of interests.

RU EN ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ ISSN 2079-3316
научная статья ПРОГРАММНОЕ И АППАРАТНОЕ ОБЕСПЕЧЕНИЕ РАСПРЕДЕЛЕННЫХ И СУПЕРКОМПЬЮТЕРНЫХ СИСТЕМ

УДК 519.687.1:004.75
10.25209/2079-3316-2026-17-3-275-314

Компактная MILP-модель для планирования многостадийных графов вычислений с однородными зависимостями в облаке

Юрий Дмитриевич **Семенов**[1✉], Олег Викторович **Сухорослов**[2]
[1] Высшая школа экономики, Москва, Россия
[2] Институт проблем передачи информации им. А. А. Харкевича РАН, Москва, Россия
[1✉] *ydsemenov@hse.ru*

Аннотация.

В статье рассматривается задача планирования ориентированных ациклических графов (DAG) вычислительных задач в облачной среде с целью минимизации длительности расписания или стоимости выполнения. Важным направлением исследований, связанных с решением сложных задач планирования, является разработка моделей смешанно-целочисленного линейного программирования (MILP), которые могут использоваться с точными решателями или служить основой для построения линейных релаксаций. Существующие MILP-модели для планирования DAG в облаке содержат большое число переменных и ограничений и известны как чрезвычайно трудные для решения. Хотя в общем случае улучить такие модели сложно, графы, встречающиеся на практике, часто обладают выраженными структурными свойствами, которые можно использовать для построения более эффективных моделей.

В работе такие свойства используются для разработки более компактной и лучше поддающейся решению MILP-модели для графов, состоящих из нескольких стадий (слоёв) с однородными зависимостями по данным. Преимущества предложенной модели демонстрируются с помощью вычислительных экспериментов.

(*Связанные тексты статьи на английском и на русском языках*)

Ключевые слова и фразы: планирование графов вычислений, облачные вычисления, оптимизационное моделирование, смешанно-целочисленное линейное программирование

Благодарности: Работа выполнена в рамках государственного задания ИППИ РАН, утвержденного Минобрнауки России.

Для цитирования: Семенов Ю. Д., Сухорослов О. В. *Компактная MILP-модель для планирования многостадийных графов вычислений с однородными зависимостями в облаке* // Программные системы: теория и приложения. 2026. **Т. 17**. № 3(72). С. 275–314. (*Англ.+русс.*) https://psta.psiras.ru/read/psta2026_3_275-314.pdf

© Семенов Ю. Д., Сухорослов О. В. 2026

1. Введение

Параллельные и распределённые приложения часто моделируются как направленные ациклические графы (DAG) вычислительных задач с зависимостями по данным между ними. Такие приложения возникают в научных вычислениях [1], распределённом машинном обучении [2] и больших данных [3]. Задача планирования графа заключается в определении для каждой задачи машины, на которой она будет выполняться, и времени начала выполнения. Традиционно алгоритмы планирования графа были сосредоточены на минимизации времени выполнения графа. Однако если граф выполняется в облаке, а не в локальном кластере, возникает другая важная метрика: стоимость выполнения. Хотя заманчиво оптимизировать оба критерия одновременно, такой подход обычно даёт несколько несравнимых Парето-эффективных расписаний, и выбор окончательного расписания требует вмешательства лица, принимающего решения, что делает его непригодным для автономных систем планирования. Поэтому мы рассматриваем однокритериальный случай, когда одна метрика оптимизируется при ограничении на другую метрику (бюджет или срок выполнения) или даже вообще без учёта другой метрики.

Основным вкладом этой статьи является улучшенная MILP-модель для частного случая многостадийных графов с однородными зависимостями по данным. Как будет показано далее, этот случай имеет определённую практическую значимость. Модель сравнивается с общей MILP-моделью для планирования графов, представленной в [4].

2. Постановка задачи

2.1. Модель графа задач

Направленный ациклический граф $G = (T, D)$ состоит из множества задач T (вершин графа) и множества зависимостей по данным D (рёбер графа). Для каждой задачи t известно время её выполнения на каждой машине v, обозначаемое как $p_{t,v}$. Обычно время выполнения получается делением размера задачи во флопсах на скорость машины. Однако мы используем более общую модель, поскольку хотим, чтобы наша модель была применима к более сложным сценариям, когда время выполнения не масштабируется линейно с производительностью машины (например, задачи, ограниченные вводом-выводом).

Рёбра графа соответствуют зависимостям по данным между задачами. Каждое ребро представляется тройкой (i, j, w), означающей, что задача i требует w байт данных, произведённых задачей i. Естественно, задача не может начать выполнение, пока не получит все необходимые данные. Если задача, произведшая данные, была выполнена на другой машине, данные должны быть переданы по сети, что вызывает коммуникационную задержку. Мы называем две задачи независимыми, если между соответствующими вершинами в графе нет направленного пути ни в одном направлении.

На практике некоторые задачи могут получать внешние входные данные и отправлять выходные данные во внешнее хранилище. В существующих исследованиях это обычно включается в модель путём добавления одной входной задачи, которая распределяет входные данные, и одной выходной задачи, которая собирает выходные данные. Мы следуем тому же подходу.

2.2. Многостадийные графы с однородными зависимостями

Ниже рассматриваются графы с определёнными структурными свойствами. Мы называем граф «многостадийным графом с однородными зависимостями», если выполняется следующее:

- Вершины графа можно разбить на k стадий $L_1, \ldots, L_k$ так, что все задачи внутри одной стадии независимы, для всех пар соседних стадий (L_i, L_{i+1}) любая задача из L_{i+1} непосредственно зависит от любой задачи из L_i, и все такие зависимости имеют одинаковый размер.
- Могут существовать зависимости между несоседней парой стадий, но в этом случае должны выполняться те же условия, что и выше (зависимости одинакового размера между всеми парами задач).

Как будет показано далее, некоторые графы, возникающие в практических приложениях, соответствуют этому определению с точностью до точной аппроксимации. Под точной аппроксимацией мы подразумеваем, что зависимости могут на самом деле не иметь одинаковый размер, но разница между ними пренебрежимо мала (например, десятки или сотни байт), что на практике не влияет на время передачи по сети.

Пример многостадийного графа с однородными зависимостями показан на рисунке 1.

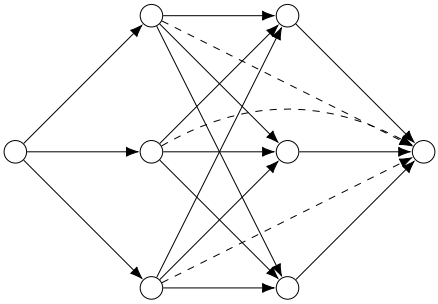


Рисунок 1. Пример многостадийного графа с однородными зависимостями. Сплошные стрелки обозначают зависимости между соседними стадиями, а пунктирные стрелки обозначают зависимости между несоседними стадиями.

2.3. Модель облачных ресурсов

Мы рассматриваем сценарий, в котором граф выполняется на наборе виртуальных машин (ВМ), арендованных у IaaS-облачного провайдера. Наиболее распространённая модель облачных ресурсов IaaS, используемая в существующих исследованиях, заключается в следующем. Пользователь имеет доступ к набору виртуальных машин V, где каждая машина принадлежит к одному из типов, предлагаемых провайдером (то есть набор машин обычно гетерогенен, но некоторые машины могут иметь эквивалентные характеристики). Каждый тип ВМ характеризуется ценой c_v и производительностью. Как мы установили ранее, мы не задаём производительность явно, а определяем её через то, насколько быстро каждая задача выполняется на данной ВМ. Мы предполагаем, что машина может выполнять одну задачу за раз, и задачи не могут быть прерваны во время выполнения.

Мы предполагаем, что передача зависимостей между задачами, выполняемыми на разных машинах, осуществляется через общее облачное хранилище (например, Amazon S3). Данные, произведённые задачей, загружаются в хранилище сразу после их вычисления и извлекаются из указанного хранилища по требованию зависимыми задачами. Передача данных размера w по сети между машиной и хранилищем требует времени w/B, где B – пропускная способность сети. Мы предполагаем, что одновременные передачи не конкурируют за пропускную способность. Наши результаты также могут быть непосредственно адаптированы к модели без общего хранилища.

Облачные провайдеры обычно предлагают ВМ на основе модели оплаты по факту использования с интервалом биллинга P в качестве параметра. Если ВМ v создана в момент времени svm_v и удалена в момент времени fvm_v, стоимость аренды машины вычисляется как

$$C_v = c_v \cdot \lceil \frac{fvm_v - svm_v}{P} \rceil. \tag{1}$$

Общая стоимость выполнения графа затем рассчитывается как сумма индивидуальных затрат на аренду ВМ. Мы не учитываем затраты на общее хранилище, поскольку для вычислительно-интенсивных графов они пренебрежимо малы по сравнению с затратами на аренду ВМ [4].

Существуют некоторые естественные ограничения, которые должны выполняться при вычислении времени создания и удаления ВМ. ВМ должна быть запущена во время всех выполнений задач, запланированных на данной ВМ, и во время всех передач данных на данную ВМ или с неё.

3. MILP модели

3.1. Базовая модель

Прежде чем представить MILP-модель для многостадийных графов с однородными зависимостями, мы опишем общую MILP-модель для планирования графов задач в облаке, представленную в [4]. Нам потребуются дополнительные обозначения.

Для простоты мы переопределяем множество зависимостей D так, чтобы оно содержало тройки $(i, j, \Delta t)$, где Δt обозначает время передачи данных в/из хранилища, вместо троек (i, j, w), где w обозначает размер данных. $\mathcal{N}$ обозначает множество всех пар (i, j) задач, между которыми нет пути в графе ни от i к j, ни наоборот.

U обозначает верхнюю границу времени выполнения графа, вычисляемую как сумма по всем задачам максимального времени выполнения задачи и максимального времени передачи данных по её исходящим зависимостям, включая как загрузку в общее хранилище, так и скачивание из него. Эта граница намеренно выбрана с большим запасом надёжности, так что её достаточно для всех ограничений big-M, используемых ниже. Наконец, ε обозначает очень малое положительное число, используемое для преобразования строгого неравенства в нестрогое.

Для простоты мы предполагаем, что граф имеет единственную выходную задачу, обозначаемую *out*. Если это не так, то можно ввести новую задачу размера 0 и добавить зависимости нулевого веса от всех выходных задач к этой новой задаче.

Модель содержит следующие переменные:

$x_{i,v} \in \{0, 1\}$ равно 1 тогда и только тогда, когда задача i размещена на ВМ v.

$st_i \in \mathbb{R}_+$ обозначает время начала выполнения задачи i.

$ft_i \in \mathbb{R}_+$ обозначает время завершения выполнения задачи i.

$\omega_{i,j} \in \{0, 1\}$ фиксирует перекрытие между независимыми задачами i и j. Если выполнения этих двух задач перекрываются, то и $\omega_{i,j}$, и $\omega_{j,i}$ устанавливаются в 1.

$svm_v \in \mathbb{R}_+$ обозначает время начала работы ВМ v.

$fvm_v \in \mathbb{R}_+$ обозначает время завершения работы ВМ v.

$I_v \in \mathbb{Z}_+$ обозначает число интервалов биллинга, использованных ВМ v.

$M \in \mathbb{R}_+$ обозначает время выполнения графа.

$C \in \mathbb{R}_+$ обозначает стоимость выполнения графа.

Модель содержит следующие ограничения:

$$\sum_{v \in V} x_{i,v} = 1 \qquad \forall i \in T \tag{2}$$

$$st_i + \sum_{v \in V} p_{i,v} \cdot x_{i,v} = ft_i \qquad \forall i \in T \tag{3}$$

$$ft_j - st_i \leqslant U \cdot \omega_{i,j} \qquad \forall (i,j) \in \mathcal{N} \tag{4}$$

$$\omega_{i,j} + \omega_{j,i} + x_{i,v} + x_{j,v} \leqslant 3 \qquad \forall (i,j) \in \mathcal{N}\ \forall v \in V \tag{5}$$

$$svm_v \leqslant fvm_v \qquad \forall v \in V \tag{6}$$

$$svm_v \leqslant st_i + U \cdot (1 - x_{i,v}) \qquad \forall v \in V\ \forall t \in T \tag{7}$$

$$fvm_v + U \cdot (1 - x_{i,v}) \geqslant ft_i \qquad \forall v \in V\ \forall i \in T \tag{8}$$

$$I_v \cdot P \geqslant fvm_v - svm_v \qquad \forall v \in V \tag{9}$$

$$(I_v - 1) \cdot P \leqslant fvm_v - svm_v - \varepsilon \qquad \forall v \in V \tag{10}$$

$$M = ft_{out} \tag{11}$$

$$C = \sum_{v \in V} c_v \cdot I_v \tag{12}$$

$$st_s \geqslant ft_p + 2\Delta t \cdot (x_{p,v} - x_{s,v}) \qquad \forall v \in V\ \forall (p,s,\Delta t) \in D \tag{13}$$

$$svm_v \leqslant st_s + U \cdot (1 - x_{s,v}) - \Delta t \cdot (x_{s,v} - x_{p,v}) \qquad \forall v \in V\ \forall (p,s,\Delta t) \in D \tag{14}$$

$$fvm_v + U \cdot (1 - x_{p,v}) \geqslant ft_p + \Delta t \cdot (x_{p,v} - x_{s,v}) \qquad \forall v \in V\ \forall (p,s,\Delta t) \in D \tag{15}$$

Ограничение (2) гарантирует, что каждая задача выполняется ровно на одной ВМ. Ограничение (3) связывает время завершения задач с временем их начала. Ограничения (4) и (5) обеспечивают корректность переменных перекрытия. Ограничения (6)–(8) устанавливают время запуска и завершения ВМ, не включая передачи данных. Ограничения (9) и (10) обеспечивают корректный подсчёт количества биллинговых интервалов. Ограничения (11) и (12) определяют время и стоимость выполнения соответственно. Ограничения, учитывающие передачи данных, различаются для каждой стратегии передачи данных. Ограничение (13) гарантирует, что никакая задача не начинает выполнение до получения всех данных от своих родительских задач. Ограничение (14) гарантирует, что все передачи данных на ВМ лежат в пределах времени её жизни. Ограничение (15) делает то же самое, но для передач с ВМ. Наконец, целевые функции, а также ограничения на бюджет и срок выполнения тривиально выражаются с использованием переменных C и M.

Переменные и ограничения, представленные выше, могут также использоваться для построения классической модели планирования графа (только с минимизацией времени выполнения). Для этого следует удалить переменные svm_v, fvm_v, I_v, C. Что касается ограничений, следует сохранить 2-5, 11 и 13.

3.2. Модель для многостадийных графов

В дальнейшем мы предполагаем, что на множестве ВМ V задан произвольный линейный порядок. ВМ сравниваются согласно этому порядку и нумеруются в соответствии с ним числами $1, \ldots, |V|$. Допуская некоторую вольность в обозначениях, мы иногда будем отождествлять ВМ из множества V с индексами $1, \ldots, |V|$.

Ключевое наблюдение, позволяющее построить более компактные модели для многостадийных графов с однородными зависимостями, заключается в следующем.

Лемма 1. *Для любого корректного расписания выполняются следующие свойства:*

1. *Если $i < i'$, то каждая задача $j \in L_i$ завершается не позже, чем начинается каждая задача $j' \in L_{i'}$.*
2. *Если задачи j и j' стадии L_i выполняются на ВМ v без других задач между ними, всегда возможно и оптимально начать j' сразу после завершения j.*
3. *Предположим, что задача j стадии L_i начинается на ВМ v в момент времени t. Если задача $j' \in L_i$ начинается сразу после j, их можно поменять местами так, что j' начнётся в момент t, а j – сразу после j'. Эта операция сохраняет корректность расписания и не изменяет ни время выполнения, ни стоимость.*

Доказательство. *1*. По определению, между всеми парами задач для всех пар соседних стадий существуют зависимости. Это означает, что если $i < i'$, каждая задача из $L_{i'}$ прямо или косвенно зависит от каждой задачи из L_i, следовательно, каждая задача из $L_{i'}$ не может начаться, пока все задачи из L_i не завершены.

2. По определению, задачи j и j', принадлежащие одной стадии L_i, имеют одно и то же множество предшественников, и для каждого предшественника размер данных, отправляемых в j и j', предполагается одинаковым. Это означает, что если задача j получила все необходимые данные к моменту t, то задача j' также может получить свои данные. Следовательно, j' может начаться сразу после j. Устранение промежутка между задачами не может продлить время жизни какой-либо ВМ, поэтому ни время выполнения, ни стоимость не увеличиваются.

3. По той же логике, что и выше, обе задачи j и j' получили все необходимые данные к моменту t, поэтому обе могут начаться в момент t. Что касается задач, зависящих от j и j', каждая такая задача зависит от обеих j и j', и зависимости имеют одинаковый размер. Поскольку в обоих случаях последняя из j и j' завершается в момент $t + p_{j,v} + p_{j',v}$, перестановка не влияет на зависимые задачи. Время выполнения и стоимость остаются прежними, поскольку время жизни ни одной ВМ не затрагивается перестановкой. □

Лемма, по существу, утверждает, что почти для всех целей нам достаточно рассматривать лишь временные отрезки $[l_{i,v}, r_{i,v}]$, в течение которых задачи стадии i выполняются на ВМ v. Внутри отрезка задачи выполняются без простоев, и точный порядок задач не имеет значения, поскольку их можно свободно менять местами. Используя эту идею, мы строим нашу модель на основе следующих переменных:

$x_{j,v} \in \{0, 1\}$ равно 1 тогда и только тогда, когда задача j размещена на ВМ v.

$l_{i,v} \in \mathbb{R}_+$ обозначает время начала стадии i на ВМ v.

$r_{i,v} \in \mathbb{R}_+$ обозначает время завершения стадии i на ВМ v. Если никакая задача стадии i не выполняется на v, $r_{i,v} = l_{i,v}$.

$r^{<}_{i,v} \in \mathbb{R}$ обозначает максимальное время завершения стадии i на любой ВМ $v' < v$ или некоторое малое отрицательное число, если ни одна ВМ $v' < v$ не выполняет стадию i.

$r^{>}_{i,v} \in \mathbb{R}$ обозначает максимальное время завершения стадии i на любой ВМ $v' > v$ или некоторое малое отрицательное число, если ни одна ВМ $v' > v$ не выполняет стадию i.

$h_{i,v} \in \{0, 1\}$ равно 1 тогда и только тогда, когда хотя бы одна задача из L_i размещена на ВМ v.

$\widehat{h}_{i,v} \in \{0, 1\}$ равно 1 тогда и только тогда, когда хотя бы одна задача из L_i не размещена на ВМ v.

$svm_v \in \mathbb{R}_+$ обозначает время начала работы ВМ v.

$fvm_v \in \mathbb{R}_+$ обозначает время завершения работы ВМ v.

$I_v \in \mathbb{Z}_+$ обозначает число интервалов биллинга, использованных ВМ v.

$M \in \mathbb{R}_+$ обозначает время выполнения графа.

$C \in \mathbb{R}_+$ обозначает стоимость выполнения графа.

В дальнейшем для заданной стадии i через S_i мы обозначаем множество пар $(i', \Delta t)$ таких, что существуют рёбра от всех задач из L_i ко всем задачам из $L_{i'}$ с временем передачи Δt. Аналогично, множество пар

$(i', \Delta t)$ таких, что существуют рёбра от всех задач из $L_{i'}$ ко всем задачам из L_i с временем передачи Δt, мы обозначаем через P_i. Этапы нумеруются от 1 до k, а $[i, k]$ обозначает числа $i, i+1, \ldots, k$. U обозначает ту же верхнюю границу времени выполнения, что и в базовой модели. Модель имеет следующие ограничения:

$$\sum_{v \in V} x_{j,v} = 1 \qquad \forall j \in T \tag{16}$$

$$l_{i,v} \geqslant r_{i-1,v} \qquad \forall i \in [2, k]\ \forall v \in V \tag{17}$$

$$l_{i,v} + \sum_{j \in L_i} p_{j,v} x_{j,v} = r_{i,v} \qquad \forall i \in [1, k]\ \forall v \in V \tag{18}$$

$$h_{i,v} \leqslant \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1, k]\ \forall v \in V \tag{19}$$

$$|L_i| \cdot h_{i,v} \geqslant \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1, k]\ \forall v \in V \tag{20}$$

$$r^{<}_{i,v} \geqslant r^{<}_{i,v-1} \qquad \forall i \in [1, k]\ \forall v \in V, v > 1 \tag{21}$$

$$r^{<}_{i,v} \geqslant r_{i,v-1} - U \cdot (1 - h_{i,v-1}) \qquad \forall i \in [1, k]\ \forall v \in V, v > 1 \tag{22}$$

$$r^{>}_{i,v} \geqslant r^{>}_{i,v+1} \qquad \forall i \in [1, k]\ \forall v \in V, v < |V| \tag{23}$$

$$r^{>}_{i,v} \geqslant r_{i,v+1} - U \cdot (1 - h_{i,v+1}) \qquad \forall i \in [1, k]\ \forall v \in V, v < |V| \tag{24}$$

$$l_{i,v} + U \cdot (1 - h_{i,v}) \geqslant r^{<}_{i',v} + 2\Delta t \qquad \forall i \in [1, k]\ \forall (i', \Delta t) \in P_i\ \forall v \in V \tag{25}$$

$$l_{i,v} + U \cdot (1 - h_{i,v}) \geqslant r^{>}_{i',v} + 2\Delta t \qquad \forall i \in [1, k]\ \forall (i', \Delta t) \in P_i\ \forall v \in V \tag{26}$$

$$\widehat{h}_{i,v} \leqslant |L_i| - \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1, k]\ \forall v \in V \tag{27}$$

$$|L_i| \cdot \widehat{h}_{i,v} \geqslant |L_i| - \sum_{j \in L_i} x_{j,v} \qquad \forall i \in [1, k]\ \forall v \in V \tag{28}$$

$$svm_v \leqslant l_{1,v} \qquad \forall v \in V \tag{29}$$

$$fvm_v \geqslant r_{k,v} \qquad \forall v \in V \tag{30}$$

$$svm_v \leqslant l_{i,v} - \Delta t \cdot (h_{i,v} + \widehat{h}_{i',v} - 1) \qquad \forall i \in [1, k]\ \forall (i', \Delta t) \in P_i\ \forall v \in V \tag{31}$$

$$fvm_v \geqslant r_{i,v} + \Delta t \cdot (h_{i,v} + \widehat{h}_{i',v} - 1) \quad \forall i \in [1,k]\ \forall (i', \Delta t) \in S_i\ \forall v \in V \tag{32}$$

$$I_v \cdot P \geqslant fvm_v - svm_v \qquad \forall v \in V \tag{33}$$

$$(I_v - 1) \cdot P \leqslant fvm_v - svm_v - \varepsilon \qquad \forall v \in V \tag{34}$$

$$M \geqslant r_{k,v} \qquad \forall v \in V \tag{35}$$

$$C = \sum_{v \in V} c_v \cdot I_v \tag{36}$$

Ограничение (16) гарантирует, что каждая задача выполняется ровно на одной ВМ. Ограничение (17) гарантирует, что каждая последующая стадия выполняется после предыдущей. Ограничение (18) связывает время начала и завершения для каждой стадии. Если ни одна задача стадии i не выполняется на ВМ v, имеем $l_{i,v} = r_{i,v}$.

Ограничения (19) и (20) определяют переменные $h_{i,v}$. Ограничения (21)-(24) определяют переменные $r^{<}_{i,v}$ и $r^{>}_{i,v}$. Они работают путём вычисления текущих максимумов $r_{i,v}$ для v, выполняющих хотя бы одну задачу стадии i, по префиксам и суффиксам V.

Ограничения (25) и (26) гарантируют, что каждая задача получает все необходимые данные перед выполнением. Ограничения (27) и (28) определяют переменные $\widehat{h}_{i,v}$.

Ограничения (29) и (30) гарантируют, что время жизни ВМ включает все задачи, выполняемые на этой ВМ. В сочетании с (17) они дополнительно гарантируют, что $svm_v \leqslant fvm_v$.

Ограничение (31) гарантирует, что все передачи данных на ВМ лежат в пределах времени её жизни. Ограничение (32) делает то же самое, но для передач с ВМ.

Ограничения (33) и (34) обеспечивают корректный подсчёт интервалов биллинга. Наконец, ограничения (35) и (36) определяют время и стоимость выполнения графа соответственно.

3.3. Сравнение моделей

Рассмотрим граф с k стадиями и l парами стадий, между которыми есть рёбра. На практике $k \ll |T|$ и $l \ll |D|$. Базовая модель содержит $O(|T||V| + |T|^2)$ целочисленных переменных и $O(|V||T|^2 + |V||D|)$ ограничений. Заметим, что $|T|^2$ – это граница размера множества $\mathcal{N}$ пар независимых задач, и в общем случае $|T|^2$ может быть чрезмерно пессимистичной границей (в качестве экстремального примера рассмотрим граф-цепочку, у которого $\mathcal{N} = \varnothing$). Однако для графов, встречающихся на практике, размер $\mathcal{N}$ часто составляет $\Theta(|T|^2)$.

В отличие от базовой модели, наша модель имеет $O(|T||V|)$ целочисленных переменных и $O(|T| + (k+l)|V|)$ ограничений.

Ещё одной важной метрикой является количество ограничений big-M, поскольку известно, что они негативно влияют на производительность решателя. Базовая модель имеет $O(|T|^2 + |V|(|T| + |D|))$ ограничений big-M, в то время как многостадийная модель имеет лишь $O((k+l)|V|)$.

4. Экспериментальное сравнение моделей

4.1. Данные

Широко используемым набором данных в современных исследованиях по планированию графов задач является WfInstances [1]. WfInstances содержит графы, собранные из 9 реальных распределённых приложений для научных вычислений. Хотя графы в этом наборе данных не соответствуют в точности нашему определению многостадийного графа с однородными зависимостями, некоторые из них соответствуют ему приближённо. Под «приближённо» мы подразумеваем следующее:

(1) Не все рёбра между парой стадий в точности равны, но время передачи для разных рёбер отличается не более чем на 10^{-3} секунды (при типичной пропускной способности облачной сети 5 Гбит/с).

В этом случае мы можем положить все времена передачи равными их среднему значению ценой небольшой погрешности, которой на практике можно пренебречь (в худшем случае погрешность времени выполнения составит не более $10^{-3}n$ секунд). Следует отметить, что использование среднего времени передачи в некоторых редких случаях может сделать результирующее расписание недопустимым.

Эту проблему можно смягчить, применив более консервативный подход и установив все времена передачи равными их максимуму. Однако это оказывает пренебрежимо малое влияние на результирующую MILP-модель, и в экспериментах ниже мы используем среднее время передачи. Для приложений, требующих гарантированной допустимости относительно исходного графа, вместо этого можно использовать максимальное время передачи без изменения формулировки.

(2) Может существовать такая пара стадий, что некоторые задачи непосредственно зависят друг от друга, а некоторые – нет, но в этом случае одна из стадий является либо первой стадией, состоящей из единственной входной задачи, либо последней стадией, состоящей из единственной выходной задачи, и существующие зависимости имеют почти нулевое время передачи (не более 10^{-3} секунды).

В этом случае мы можем добавить недостающие рёбра с нулевым временем передачи. Идея, лежащая в основе этой процедуры, состоит в том, что задачи, которые не получают внешние входные данные или не отправляют выходные данные, всё равно сообщают свой статус планировщику, и это можно рассматривать как внешние входные/выходные данные пренебрежимо малого размера.

Таким образом, добавление дополнительных рёбер нулевого размера от входной задачи или к выходной задаче не делает модель нереалистичной. Пример показан на рисунке 2. Некоторая стадия может лишь частично

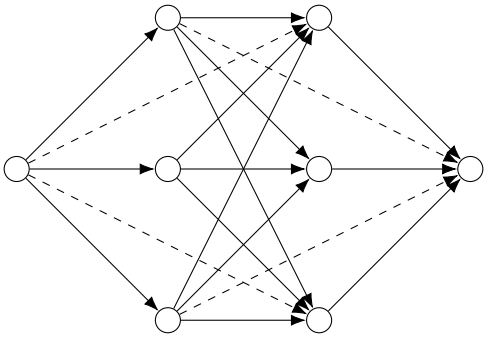


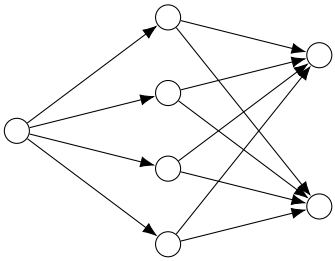
Рисунок 2. Граф, приближённо удовлетворяющий определению многостадийного графа с однородными зависимостями

зависеть от входной стадии, и выходная стадия может лишь частично зависеть от некоторой стадии, но это не страшно, если эти зависимости очень малого размера.

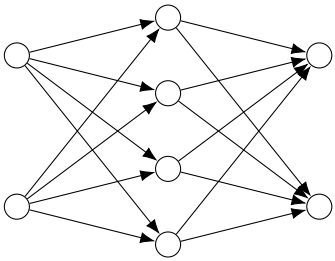
Анализируя набор данных WfInstances, мы обнаружили следующие приложения, которые приближённо соответствуют определению многостадийного графа с однородными зависимостями:

- **BLAST.** Графы этого биоинформатического приложения имеют простую трёхстадийную структуру fork-join.
- **BWA.** Аналогично BLAST, но первая стадия имеет две задачи вместо одной.
- **Seismology.** Эти графы имеют простую двухстадийную структуру: первая стадия состоит из параллельных задач, а вторая стадия представляет собой всего одну задачу, которая обрабатывает и объединяет результаты предыдущих задач.

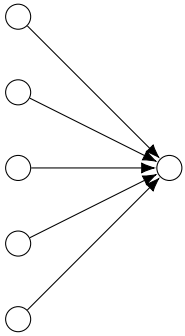
Примеры, показывающие структуру графов BLAST, BWA и Seismology, приведены на рисунке 3. Для наглядности графы изображены без искусственных входной и выходной задач.



(*а*) BLAST



(*б*) BWA



(*в*) Seismology

Рисунок 3. Примеры структур графов

4.2. Описание эксперимента

Мы сравниваем производительность двух MILP-моделей на графах из WfInstances для трёх упомянутых выше приложений. Мы ограничиваемся графами, содержащими не более 110 задач, включая дополнительные входную и выходную задачи. Следуя [4], мы рассматриваем набор ВМ из 4 самых дешёвых типов ВМ в каждом из семейств Amazon EC2 m5 и c5, по 3 ВМ каждого типа (всего 24 ВМ). Интервал биллинга составляет 1 секунду.

Модели решались с помощью MILP-решателя CPLEX 20.1 на процессоре Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz со 128 ГБ оперативной памяти. Решателю было разрешено использовать до 32 параллельных потоков. Ограничение по времени на решение модели составляло 1 час.

Мы тестируем три постановки задачи: минимизация времени выполнения без дополнительных ограничений, минимизация стоимости при ограничении на срок выполнения и минимизация времени выполнения при ограничении на бюджет. Последние две постановки требуют задания срока выполнения или бюджета.

Стандартный подход [4–6] состоит в том, чтобы взять некоторую грубую оценку оптимального времени выполнения или стоимости и преобразовать её в ограничение путём умножения на некоторый заранее определённый параметр α. Оценка стоимости вычислялась путём последовательного запуска всех задач на самом дешёвом типе ВМ, как в [4, 5]. Оценка времени выполнения вычислялась путём запуска каждой задачи на отдельном экземпляре самого быстрого типа ВМ в предположении, что передачи данных происходят мгновенно, как в [4, 6].

Параметр α был установлен равным 10 для срока выполнения и 2 для бюджета. Причина использования разных α для разных ограничений заключается в том, что слишком малое значение α сделало бы задачу тривиально недопустимой, а слишком большое значение α сделало бы ограничение тривиально выполнимым и, следовательно, бесполезным. Что является слишком малым или слишком большим, зависит от качества оценки, которое различается для времени и стоимости, поэтому и параметры разные.

4.3. Результаты

В таблице 1 показано количество целочисленных переменных и ограничений в MILP-моделях для задач без ограничений и с ограничениями на срок выполнения/бюджет соответственно (заметим, что минимизация

Таблица 1. Размеры моделей в количествах целочисленных переменных и ограничений для минимизации времени выполнения

Графы	Размер модели			
	общей		многостадийной	
	$I^{(1)}$	ограничений	$I^{(1)}$	ограничений
без ограничений на срок выполнения или бюджет				
blast-chameleon-large-*	13545	257480	3231	1573
blast-chameleon-small-*	3105	43100	1791	1513
bwa-chameleon-small-*	13780	259956	3255	1574
seismology-chameleon-100p-*	13081	252554	3041	1239
при ограничеиях на срок выполнения или бюджет				
blast-chameleon-large-*	13570	277042	3256	1959
blast-chameleon-small-*	3130	51142	1816	1899
bwa-chameleon-small-*	13805	284366	3280	1960
seismology-chameleon-100p-*	13106	262372	3066	1529

(1) количество целочисленных переменных

стоимости при ограничении на срок выполнения и минимизация времени выполнения при ограничении на бюджет порождают MILP-модели одинакового размера).

В таблицах 2, 3 и 4 представлены время решения модели, относительный зазор решателя и статус решателя для минимизации времени выполнения без ограничений, минимизации стоимости при ограничении на срок выполнения и минимизации времени выполнения при ограничении на бюджет соответственно.

Зазор решателя вычисляется как $\frac{best-bound}{best} \cdot 100\%$, где $best$ – лучшее расписание, найденное решателем, а $bound$ – лучшая нижняя граница оптимального значения целевой функции, найденная решателем. Если модель решена до оптимальности], то зазор равен 0. Если модель не нашла ни одного допустимого решения, то зазор равен ∞. Если зазор равен ∞ и решатель завершил работу до достижения лимита времени, то это означает, что задача доказуемо недопустима.

Статус решателя – одно из следующего:

«optimal», если модель была решена до оптимальности;

«feasible», если было найдено некоторое допустимое решение, но его оптимальность не доказана;

«infeasible», если решатель доказал, что допустимого решения не существует; и

«unknown», если допустимое решение не было найдено, но и недопустимость также не удалось доказать.

Решателю не удалось решить базовую MILP-модель до оптимальности ни для одной задачи ни на одном графе в течение 1 часа. Многостадийная модель была решена на графах *blast-chameleon-small-** (за несколькими исключениями), на графе Seismology (кроме минимизации времени исполнения) и на двух графах BWA в случае минимизации стоимости при ограничении на срок выполнения.

Несмотря на то, что в большинстве случаев многостадийную модель не удаётся решить до оптимальности в течение 1 часа, для графов BLAST и Seismology она даёт решения с зазором оптимальности ниже 4% (за единственным исключением в лице оптимизации при ограничении на бюджет для *blast-chameleon-small-001*), если задача допустима (и во многих случаях зазор составляет менее 1%). Её производительность на графах BWA несколько ухудшается, но она всё равно даёт меньший зазор, чем базовая MILP-модель.

Базовая MILP-модель, с другой стороны, часто даёт решения с зазором более 60%, а для оптимизации при ограничении на срок выполнения зазор никогда не бывает ниже 90%. Что касается размера модели, многостадийная модель достигает сокращения количества ограничений до 200 раз и сокращения количества целочисленных переменных до 4 раз.

Таблица 2. Минимизация времени выполнения без ограничений: время решения, зазор и статус решателя

граф	общая модель			многостадийная модель		
	время (с)	статус	зазор (%)	время (с)	статус	зазор (%)
blast-chameleon-large-001	3600.00	feasible	90.63	3600.00	feasible	0.25
blast-chameleon-large-002	3600.00	feasible	92.64	3600.00	feasible	0.12
blast-chameleon-large-003	3600.00	feasible	95.44	3600.00	feasible	0.24
blast-chameleon-large-004	3600.00	feasible	93.08	3600.00	feasible	0.17
blast-chameleon-large-005	3600.00	feasible	93.37	3600.00	feasible	0.17
blast-chameleon-small-001	3600.00	feasible	80.22	3600.00	feasible	3.09
blast-chameleon-small-002	3600.00	feasible	71.92	20.40	optimal	0.00
blast-chameleon-small-003	3600.00	feasible	69.96	1578.76	optimal	0.00
blast-chameleon-small-004	3600.00	feasible	71.43	638.41	optimal	0.00
blast-chameleon-small-005	3600.00	feasible	71.31	42.10	optimal	0.00
bwa-chameleon-small-001	3600.00	feasible	19.23	3600.00	feasible	4.81
bwa-chameleon-small-002	3600.00	feasible	21.95	3600.00	feasible	9.91
bwa-chameleon-small-003	3600.00	feasible	23.34	3600.00	feasible	15.50
bwa-chameleon-small-004	3600.00	feasible	21.29	3600.00	feasible	11.50
bwa-chameleon-small-005	3600.00	feasible	21.04	3600.00	feasible	7.49
seismology-chameleon-100p-001	3600.00	feasible	65.79	3600.00	feasible	0.14

Таблица 3. Минимизация стоимости с предельным сроком ($\alpha = 10$): время решения, зазор и статус решателя

граф	общая модель			многостадийная модель		
	время (с)	статус	зазор (%)	время (с)	статус	зазор (%)
blast-chameleon-large-001	3600.00	unknown	∞	3600.00	feasible	0.15
blast-chameleon-large-002	3600.00	unknown	∞	3600.00	feasible	0.07
blast-chameleon-large-003	3600.00	unknown	∞	3600.00	feasible	0.17
blast-chameleon-large-004	3600.00	unknown	∞	3600.00	feasible	0.18
blast-chameleon-large-005	3600.00	unknown	∞	3600.00	feasible	0.11
blast-chameleon-small-001	3600.00	unknown	∞	0.07	infeasible	∞
blast-chameleon-small-002	3600.00	feasible	93.22	59.95	optimal	0.00
blast-chameleon-small-003	3600.00	feasible	93.34	3222.04	optimal	0.00
blast-chameleon-small-004	3600.00	feasible	92.86	3600.00	feasible	2.34
blast-chameleon-small-005	3600.00	feasible	93.45	3600.00	feasible	0.13
bwa-chameleon-small-001	3600.00	feasible	100.00	42.41	optimal	0.00
bwa-chameleon-small-002	3600.00	feasible	100.00	3600.00	feasible	0.01
bwa-chameleon-small-003	3600.00	feasible	100.00	3600.00	feasible	1.27
bwa-chameleon-small-004	3600.00	feasible	100.00	16.40	optimal	0.00
bwa-chameleon-small-005	3600.00	feasible	100.00	3600.00	feasible	1.34
seismology-chameleon-100p-001	3600.00	feasible	100.00	75.13	optimal	0.00

Таблица 4. Минимизация времени выполнения с бюджетом ($\alpha = 2$): время решения, зазор и статус решателя

граф	общая модель			многостадийная модель		
	время (с)	статус	зазор (%)	время (с)	статус	зазор (%)
blast-chameleon-large-001	3600.00	unknown	∞	3600.00	feasible	0.52
blast-chameleon-large-002	3600.00	feasible	96.74	3600.00	feasible	0.20
blast-chameleon-large-003	3600.00	feasible	95.78	3600.00	feasible	0.24
blast-chameleon-large-004	3600.00	unknown	∞	3600.00	feasible	0.37
blast-chameleon-large-005	3600.00	unknown	∞	3600.00	feasible	0.19
blast-chameleon-small-001	3600.00	feasible	76.62	3600.00	feasible	8.76
blast-chameleon-small-002	3600.00	feasible	74.32	44.74	optimal	0.00
blast-chameleon-small-003	3600.00	feasible	72.85	2194.55	optimal	0.00
blast-chameleon-small-004	3600.00	feasible	71.63	109.33	optimal	0.00
blast-chameleon-small-005	3600.00	feasible	73.26	29.13	optimal	0.00
bwa-chameleon-small-001	3600.00	feasible	22.49	3600.00	feasible	2.54
bwa-chameleon-small-002	3600.00	feasible	72.44	3600.00	feasible	22.73
bwa-chameleon-small-003	3600.00	unknown	∞	3600.00	feasible	5.85
bwa-chameleon-small-004	3600.00	unknown	∞	3600.00	feasible	11.60
bwa-chameleon-small-005	3600.00	unknown	∞	3600.00	feasible	13.19
seismology-chameleon-100p-001	3600.00	feasible	99.58	821.60	optimal	0.00

5. Заключение

В данной работе предложена новая компактная MILP-модель для планирования многостадийных графов задач с однородными зависимостями в облаке. Эта модель значительно меньше стандартной модели благодаря сжатию стадий. Эксперименты демонстрируют преимущество предложенной модели: она существенно сокращает количество целочисленных переменных и ограничений и в большинстве случаев находит оптимальные или близкие к оптимальным допустимые решения в пределах 1-часового лимита времени. Перспективным направлением будущих исследований является усиление формулировки для нахождения оптимальных решений в большем числе случаев.

Список использованных источников

[1] Coleman T., Casanova H., Pottier L., Kaushik M., Deelman E., Ferreira da Silva R. *Wfcommons: A framework for enabling scientific workflow research and development* // Future Generation Computer Systems.– March 2022.– Vol. **128**.– Pp. 16–27. ↑296, 305

[2] Mayer R., Mayer Ch., Laich L. *The tensorflow partitioning and scheduling problem: it's the critical path!Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*, DIDL'17 (Las Vegas, Nevada, USA, December 11–15, 2017), New York: ACM.– 2017.– ISBN 978-1-4503-5169-0.– Pp. 1–6. ↑296

[3] Aseman-Manzar M.-M., Karimian-Aliabadi S., Entezari-Maleki R., Egger B., Movaghar A. *Cost-aware resource recommendation for DAG-based big data workflows: An apache spark case study* // IEEE Transactions on Services Computing.– 2023.– Vol. **16**.– No. 3.– Pp. 1726–1737. ↑296

[4] Semenov Yu., Sukhoroslov O. *MIP models and complexity results for DAG scheduling in the cloud* // *Mathematical Optimization Theory and Operations Research*, MOTOR 2025 (Novosibirsk, Russia, July 7–11, 2025), Lecture Notes in Computer Science.– vol. **15681**, eds. Kochetov Yu., Khachay M., Eremeev A., Pardalos P., Cham: Springer.– 2025.– ISBN 978-3-031-97076-4.– Pp. 317–331. ↑296, 298, 299, 307, 308

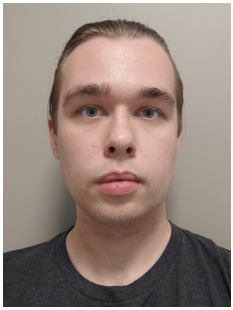
[5] Li H., Wang D., Cañizares Abreu J. R., Zhao Q., Bonilla Pineda O. *PSO+LOA: hybrid constrained optimization for scheduling scientific workflows in the cloud* // The Journal of Supercomputing.– 2021.– Vol. **77**.– No. 11.– Pp. 13139–13165. ↑308

[6] Shobeiri P., Rastaghi M. A., Abrishami S., Shobiri B. *PCP-ACO: a hybrid deadline-constrained workflow scheduling algorithm for cloud environment* // The Journal of Supercomputing.– 2024.– Vol. **80**.– No. 6.– Pp. 7750–7780. ↑308

Поступила в редакцию 29.05.2026;
одобрена после рецензирования 29.05.2026;
принята к публикации 28.07.2026;
опубликована онлайн 21.09.2026.

Рекомендовал к публикации *д.ф.-м.н. С. М. Абрамов*

Информация об авторах:



Юрий Дмитриевич Семенов

Аспирант НИУ ВШЭ. Научные интересы включают распределённые вычисления и комбинаторную оптимизацию.

0009-0004-8996-4857

e-mail: *ydsemenov@hse.ru*



Олег Викторович Сухорослов

Кандидат технических наук, старший научный сотрудник ИППИ РАН, доцент НИУ ВШЭ. Научные интересы: распределённые вычисления, облачные вычисления, управление ресурсами, планирование и имитационное моделирование.

0000-0003-2836-2524

e-mail: *sukhoroslov@iitp.ru*

Вклад авторов: *Ю. Д. Семенов* – 80% (методология, программное обеспечение, валидация, расследование, написание черновой версии); *О. В. Сухорослов* – 20% (сбор материала, доработка и редактирование, администрирование).

Декларация об отсутствии личной заинтересованности: *благополучие авторов не зависит от результатов исследования.*