



PDF Download
3754598.3754673.pdf
14 January 2026
Total Citations: 0
Total Downloads: 49

Latest updates: <https://dl.acm.org/doi/10.1145/3754598.3754673>

RESEARCH-ARTICLE

Carbon-Aware Workflow Scheduling with Fixed Mapping and Deadline Constraint

DOMINIK SCHWEISGUT, Humboldt University of Berlin, Berlin, Germany

A. BENOIT, Georgia Institute of Technology, Atlanta, GA, United States

YVES L ROBERT, Lyon Higher Normal School, Lyon, Auvergne-Rhone-Alpes, France

HENNING MEYERHENKE, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

Open Access Support provided by:

Karlsruhe Institute of Technology

Georgia Institute of Technology

Lyon Higher Normal School

Humboldt University of Berlin

Published: 08 September 2025

[Citation in BibTeX format](#)

ICPP '25: 54th International Conference on Parallel Processing
September 8 - 11, 2025
CA, San Diego, USA

Carbon-Aware Workflow Scheduling with Fixed Mapping and Deadline Constraint

Dominik Schweisgut
Humboldt-Universität zu Berlin
Berlin, Germany
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
dominik.schweisgut@hu-berlin.de

Anne Benoit
ENS Lyon
Lyon, France
IUF
Paris, France
IDEaS, Georgia Tech
Atlanta, USA
anne.benoit@ens-lyon.fr

Yves Robert
ENS Lyon
Lyon, France
IUF
Paris, France
yves.robert@ens-lyon.fr

Henning Meyerhenke
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
meyerhenke@kit.edu

Abstract

Large data and computing centers consume a significant share of the world's energy consumption. A prominent subset of the workloads in such centers are workflows with interdependent tasks, usually represented as directed acyclic graphs (DAGs). To reduce the carbon emissions resulting from executing such workflows in centers with a mixed (renewable and non-renewable) energy supply, it is advisable to move task executions to time intervals with sufficient green energy when possible. To this end, we formalize the above problem as a scheduling problem with a given mapping and ordering of the tasks. We show that this problem can be solved in polynomial time in the uniprocessor case. For at least two processors, however, the problem becomes NP-hard. Hence, we propose a heuristic framework called CaWoSched that combines several greedy approaches with local search. To assess the 16 heuristics resulting from different combinations, we also devise a simple baseline algorithm and an exact ILP-based solution. Our experimental results show that our heuristics provide significant savings in carbon emissions compared to the baseline.

CCS Concepts

• **Theory of computation** → **Design and analysis of algorithms**.

Keywords

Workflow scheduling, carbon-aware computing, DAG scheduling, heterogeneous platforms

ACM Reference Format:

Dominik Schweisgut, Anne Benoit, Yves Robert, and Henning Meyerhenke. 2025. Carbon-Aware Workflow Scheduling with Fixed Mapping and Deadline Constraint. In *54th International Conference on Parallel Processing (ICPP '25)*, September 08–11, 2025, San Diego, CA, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3754598.3754673>

1 Introduction

The number of large and geo-distributed data and computing centers grows rapidly and so is the amount of data processed by them. Their services have become indispensable in industry and academia alike. Yet, these services result in a globally significant energy consumption as well as carbon emissions due to computations and data transfer, see for example [3] for concrete numbers. Since the combined carbon footprint of all data/computing centers on the globe is even higher than that of air traffic [24], reducing this footprint is of immense importance, both from an ecological, political, and economical perspective. On the technical side, data/computing centers have started to use a mix of different power sources, giving priority to lower carbon-emitting technologies (solar, wind, nuclear) over higher ones (coal, natural gas). This raises new challenges and opportunities for HPC (High Performance Computing) scientists; for example, designing efficient scheduling algorithms was already a complicated task when computer platforms had only a single power source – and thus the same level of carbon emissions at each point in time. This task becomes even more difficult when a mix of power sources leads to different carbon emissions over time: in addition to optimizing only standard performance-related objectives, one important new objective is to optimize the total amount of carbon emissions induced by the execution of all applications in a particular data/computing center.

Many workloads in a data/computing center, not only but in particular in a scientific context, can be seen as workflows consisting of individual tasks with input/output relations. The algorithmic task we focus on in this paper is to schedule such workflows (abstracted as directed acyclic task graphs) on a parallel platform within some



This work is licensed under a Creative Commons Attribution 4.0 International License. *ICPP '25, San Diego, CA, USA*

© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2074-1/25/09
<https://doi.org/10.1145/3754598.3754673>

deadline. To focus on carbon footprint minimization, we assume that the mapping and ordering of all tasks and communications are already given, for instance as the result of executing the de-facto standard HEFT algorithm [35]. A somewhat similar approach of assuming a fixed mapping can be found in the literature, e. g., for energy- and reliability-aware task scheduling and frequency scaling in embedded systems [28].

In our simplified setting with a given mapping, minimizing the total execution time, or makespan, can easily be achieved in linear time with the ASAP greedy algorithm: simply execute each task as soon as all preceding tasks and corresponding communications have completed. The computing platform that we target instead has a time-varying amount of green energy available, for example due to solar and/or wind power produced for its data/computing center. Moreover, the carbon emissions can vary from processor to processor due to the latter's different power demands, making the platform completely *carbon-heterogeneous*.

The problem becomes combinatorial: while we cannot change the mapping nor the ordering of the tasks on each processor, we can shift the tasks (and the corresponding communication operations) to benefit from lower-carbon intervals, while enforcing the deadline. Previous studies have shown that exploiting lower-carbon intervals can be very beneficial, see e. g., [39]. Yet, this previous line of research has either not focused on scheduling individual workflows or focused on reducing energy consumption [7, 36] rather than carbon emission. As we outline in more detail in Section 2, carbon-aware scheduling algorithms are still in their infancy.

Contributions. The main contributions of this paper are both theoretical and practical. On the theory side, we lay the foundations for the problem complexity, with (i) a sophisticated fully-polynomial time dynamic programming algorithm for the single processor case, (ii) a proof of strong NP-completeness for a simplified instance with 2 or more processors, independent tasks, and carbon-homogeneous processors, and (iii) the formulation of the general problem as an integer linear program (ILP). On the practical side, we design efficient algorithms that greatly decrease the total carbon cost compared to a standard carbon-unaware competitor; for small instances, our experimental results indicate that our algorithms achieve a quality that is close to the optimal one derived from the ILP.

Outline. The rest of the paper is organized as follows. Section 2 surveys related work. In Section 3, we detail the framework. Section 4 is devoted to complexity results. We introduce new carbon-aware algorithms in Section 5 and assess their performance through an extensive set of simulations in Section 6. Finally, we give concluding remarks and hints for future work in Section 7. Note that material omitted due to space constraints can be found in [32].

2 Related Work

Carbon-aware computing has received increasing attention in the past few years, acknowledging the clearly non-negligible share data/computing centers have on mankind's carbon footprint – as well as the need for action to reduce the emissions given the rapid increase in data to be processed [10]. Most works in this more general line of research retain a high-level workload perspective and thus do not consider the concrete task of workflow scheduling. For example, a carbon-aware load balancing algorithm to

reduce the carbon footprint of geo-distributed data centers considers abstract workloads, not interdependent tasks [26]. It uses the *alternating direction method of multipliers* to move workloads to locations with lower carbon intensity. On a similar granularity, global cloud providers use scheduler-agnostic workload shifting to less carbon-intensive data centers, depending on the projected availability of green energy in suitable locations and time intervals [29]. A high-level workload perspective and a similar objective is used by Hall et al. [18]. They devise a two-phase approach of (i) day-ahead planning based on historical data and (ii) real-time job placement and scheduling. As they consider abstract workloads and not workflows with interdependencies, their approach is not directly comparable, either. Finally, while Breukelmann et al. [9] model interconnected data centers as a weighted graph, they still consider unrelated batch compute jobs as the workload. They formalize the optimal allocation problem in this setting as a single-leader multiple-follower Stackelberg game and suggest an ad-hoc algorithm (which is not applicable in our setting) to solve it.

Regarding workflow scheduling in general, we refer the interested reader to surveys [2, 25] and a monograph [34] for a broader overview. One possible way to categorize workflow scheduling algorithms is to distinguish online algorithms (which do not know the complete workflow when taking decisions for tasks) and plan-based algorithms. Since this paper proposes a plan-based algorithm, we focus on the latter. Even rather simplistic versions of plan-based scheduling are NP-hard [15], which motivates the use of heuristics for real-world applications. Two common approaches are list- and partitioning-based heuristics. HEFT (heterogeneous earliest finish time) [35] is a very influential and still popular list-scheduling algorithm that has seen numerous extensions and variations over the years [5, 8, 30, 31, 33]. It has two main phases that (i) assign priorities to tasks and (ii) then assign tasks to processors based on the priorities from the previous phase.

Partitioning-based scheduling heuristics, in turn, group tasks into blocks and assign these blocks to processors, see e. g. [23, 27, 37]. This aggregation step helps in reducing the complexity of dealing with individual task assignments in large-scale workflows.

Two prominent algorithms for energy-efficient workflow scheduling are GreenHEFT [13] and MOHEFT [14]. Both heuristics optimize *where* tasks are scheduled in order to save energy. Similar to one type of our heuristics, TaskFlow [36] exploits *slack* in workflows, i. e., it takes advantage of tolerable delays by executing the corresponding tasks on more energy-efficient hardware. Yet, they all do not optimize for carbon emissions and thus do not consider *when* tasks should run in order to exploit green energy availability.

The importance of reducing carbon emissions has led to a number of papers working on this goal. Wen et al. [38], for example, propose a genetic algorithm for adaptive workflow mapping whose main rationale is to move tasks between geographically distributed data centers – depending on their energy mix. The approach only provides a mapping of tasks to data centers, but no task starting times. Moreover, the largest workflows in their experiments have up to 1,000 tasks, an indication that the genetic algorithm is quite time-consuming. A similar rationale of moving tasks to locations with sufficient green energy is used by Hanafy et al. [19], who scale the resources assigned to elastic cloud applications in a carbon-aware manner. Considering a single data center location, in turn, Wiesner

et al. [39] investigate how beneficial shifting of execution times to intervals with lower carbon emissions can be. By evaluating the impact of time constraints, scheduling strategies, and forecast accuracy, they find significant potential (under certain conditions) and provide guidance regarding corresponding data center policies.

Altogether, this work is in line with the general trend of minimizing energy consumption and/or carbon emissions. However, to the best of our knowledge, it is the first to focus on optimizing the scheduling of a given workflow mapping and ordering to benefit from time-varying green energy.

3 Framework

We use a suitable time unit (e.g. seconds, minutes, ...) and express all parameters as integer multiples of this unit.

Platform and application. The target platform C is a cluster of P heterogeneous processors $\{p_1, \dots, p_P\}$. The target application consists of a workflow modeled as a Directed Acyclic Graph (DAG) $G = (V, E, \omega, c)$, where the vertex set V represents the set of n tasks v_1, \dots, v_n . An edge $(v_i, v_j) \in E$ represents a precedence constraint between tasks v_i and v_j , meaning that task v_j cannot start before task v_i is completed and its output was communicated to the processor handling task v_i .

We assume that the mapping is given, as well as the ordering of the tasks and the communication operations (i.e., data transfer) on each processor. Therefore, if two tasks are mapped on the same processor with task v_i planned before task v_j , we add a precedence constraint (v_i, v_j) to E , to ensure that the order is respected.

Given the mapping, the set of communications is represented by $E' \subseteq E$, which contains all edges $(v_i, v_j) \in E$ such that the two tasks are mapped on different processors, in which case data must be communicated between both processors before v_j can start its execution. However, when the two tasks are on the same processor $((v_i, v_j) \in E \setminus E')$, task v_j can start as soon as task v_i is finished.

Each task $v_i \in V$ has a running time $\omega(v_i)$, and each edge $(v_i, v_j) \in E'$ has a communication time $c(v_i, v_j)$, which accounts for the amount of data that has to be communicated from the processor of task v_i to the processor of task v_j . Computation and communication times can be arbitrary and are given, which allows us to account for any heterogeneity in computing speeds and/or link bandwidths across the processors.

Communication-enhanced DAG G_c . For simplicity, we assume that the cluster employs a fully connected, full-duplex communication topology, where each processor can directly communicate with every other processor simultaneously in both directions. We introduce $P(P-1)$ fictional processors $\{p_{P+1}, \dots, p_{P^2}\}$, one per communication link, whose role is to execute all (potential) communications on that link. This will clarify how to compute the cost of a schedule. With these additional processors, each communication $(v_i, v_j) \in E'$ becomes a (fictional) task $v_{i,j}$ of length $\omega(v_{i,j}) = c(v_i, v_j)$. Furthermore, we add dependencies $(v_i, v_{i,j})$ and $(v_{i,j}, v_j)$, each with zero communication cost. Since the order of communications is also assumed to be given with the mapping, we add precedence constraints to express this order if two tasks $v_{i,j}$ and $v_{k,\ell}$ are on the same communication link (represented by a fictional processor). This is similar to the precedence constraints added to express the order of computing tasks and we refer to this set of constraints as E'' .

We obtain a communication-enhanced DAG $G_c = (V_c, E_c, \omega)$, where V_c contains both V and all $|E'|$ communication tasks $v_{i,j}$:

$$V_c = V \cup \{v_{i,j} \mid (v_i, v_j) \in E'\},$$

and E_c contains both the precedence relations expressing the order on each processor $(E \setminus E')$ and the new dependencies related to communication tasks:

$$E_c = (E \setminus E') \cup \{(v_i, v_{i,j}), (v_{i,j}, v_j) \mid (v_i, v_j) \in E'\} \cup E''.$$

This DAG does not have any communication costs, since they have all been replaced by tasks. The number of tasks is $N = |V_c| = n + |E'|$, the mapping of tasks on processors is given as well as the order of tasks on processors (both original tasks from V and communication tasks from E'). The construction of the extended platform with P^2 processors and the corresponding DAG G_c is straightforward.

Power profile. In every time unit, processor p_i , $1 \leq i \leq P^2$, consumes idle power of $\mathcal{P}_{\text{idle}}^i$ units, to which a working power of $\mathcal{P}_{\text{work}}^i$ units is added whenever P_i is active, for a total power $\mathcal{P}^i(t)$ at time t . A processor executing a task or a communication is active from that operation's start to its end. Note that communication processors are likely to consume much less than regular (computing) processors. In particular, we could set the static power of a link that is never used to 0.

The horizon is an interval $[0, T[$, where T is the deadline. We assume that the horizon is divided into J intervals $\{I_1, \dots, I_J\}$, where interval I_j has length ℓ_j and $\sum_{j=1}^J \ell_j = T$. We let $I_j = [b_j, e_j[$ so that $\ell_j = e_j - b_j$ for every $1 \leq j \leq J$. The set of starting and ending times of the J intervals is

$$\mathcal{E} = \{b_1 = 0, e_1 = b_2, e_2 = b_3, \dots, e_{J-1} = b_J, e_J = T\}.$$

Within each interval I_j , there is a (constant) green power budget \mathcal{G}_j for each time unit $t \in I_j$. If the power consumed by all processors at time t exceeds this budget, the platform must resort to brown carbonated power, which will incur some carbon cost at time t . This is the key hypothesis of this work: the carbon cost of a schedule will depend in the end on which intervals are heavily used, or not, by the processors. The scheduler must maximize the benefit from greener intervals while enforcing all dependencies and meeting the global deadline T .

Carbon cost. Given a *schedule*, i.e., a start time for each task of V_c (i.e., including communication tasks), it is easy to compute its total carbon cost by looping over the T time units: for each time unit t , sum up the power consumed by each processor, either computing or communicating, $\mathcal{P}_t = \sum_{i=1}^{P^2} \mathcal{P}^i(t)$ (which may include $\mathcal{P}_{\text{work}}^i$ or not). The carbon cost for $t \in I_j$ is assumed to be proportional to the non-green power, and hence we simply write $CC_t = \max(\mathcal{P}_t - \mathcal{G}_j, 0)$. The total carbon cost of the schedule is then $CC = \sum_{t=0}^{T-1} CC_t$. However, this approach has exponential (in fact, pseudo-polynomial) complexity, since the problem instance has size $O(P^2 + N + J + \log(T) + \max_{1 \leq i \leq P^2} \log(\mathcal{P}_{\text{idle}}^i) + \max_{1 \leq i \leq P^2} \log(\mathcal{P}_{\text{work}}^i) + \max_{1 \leq j \leq J} \log(\mathcal{G}_j))$. To compute the cost of a schedule in polynomial time, we need to proceed interval by interval and create sub-intervals each time a task starts or ends, so that the number of active tasks is constant within each subinterval. The carbon cost per subinterval then depends on the power cost of the subinterval (CC_t is constant within a subinterval) and the interval length. Details can be found in [32, Appendix A.1].

Optimization problem. The objective is to find an optimal schedule, defined as a schedule whose total carbon cost CC is minimum. To achieve this goal, the scheduler can shift around tasks (including communication tasks) to benefit from greener intervals, while enforcing all dependencies and meeting the deadline T .

4 Complexity Results

In this section, we present an involved dynamic programming (DP) algorithm, establishing that the problem with a single processor has polynomial time complexity. On the contrary, and as expected, the problem with several processors is strongly NP-complete, even with homogeneous processors and no communications, but we can formulate the general problem as an integer linear program (ILP).

4.1 Polynomial DP algorithm for one processor

THEOREM 4.1. *The problem instance with a single processor has polynomial time complexity.*

PROOF. Consider the problem instance with a single processor executing tasks v_1, \dots, v_n in this order (with $n = |V|$, no communication tasks in this case). We start with a pseudo-polynomial algorithm: for $1 \leq i \leq n$ and $1 \leq t \leq T$, we let $Opt(i, t)$ be the cost of an optimal schedule for the first i tasks and where task v_i completes its execution exactly at time t , storing the value ∞ if no such schedule exists. We have the induction formula

$$Opt(i, t) = \min_{s \leq t - \omega(v_i)} \{Opt(i-1, s) + cc(v_i, t)\}, \quad (1)$$

where $cc(v_i, t)$ is the cost to execute task v_i during the interval $[t - \omega(v_i), t]$. Since there is a single processor, this can be computed in linear time by computing the length of its intersection with the intervals I_j . In Eq. (1), we loop over possible termination dates for the previous task v_{i-1} . For the initialization, we simply compute the value of $Opt(1, t)$ for all $t \geq \omega(v_1)$, and let $Opt(1, t) = \infty$ for $t < \omega(v_1)$.

This dynamic programming algorithm is pseudo-polynomial because it tries all possible values $t \in [1, T]$ for the end times of the tasks. To derive a polynomial-time algorithm, we show that we can derive an optimal algorithm while restricting to a polynomial number of end dates.

Given any single processor schedule \mathcal{S} , we define a *block* as a set of consecutive tasks in the schedule, i. e., there is no idle time between tasks within a block. Note that if a task has idle time before and after it, it forms a block by itself. Furthermore, schedules where each block either starts or ends at a time in \mathcal{E} are called \mathcal{E} -schedules (recall that $\mathcal{E} = \{b_1 = 0, e_1 = b_2, e_2 = b_3, \dots, e_{J-1} = b_J, e_J = T\}$ is the set of starting/ending times of intervals). We can then prove the following Lemma (see proof in [32, Appendix A.2]) \square

LEMMA 4.2. *With a single processor, there always exists an optimal \mathcal{E} -schedule.*

According to this lemma, we can therefore restrict the pseudo-polynomial dynamic programming algorithm to only using task end times that belong to a refined set of end times \mathcal{E}' , which is of size $O(n^3 J)$ (see [32, Appendix A.2]) thereby leading to a fully polynomial running time.

4.2 NP-completeness of the multiprocessor case

THEOREM 4.3. *The problem instance with several processors is strongly NP-complete, even with uniform processors and independent tasks (hence no communications).*

PROOF. We consider the class UCAS of decision problem instances with P processors with uniform power consumption, i.e., $\mathcal{P}_{\text{idle}}^i = 0$, $\mathcal{P}_{\text{work}}^i = 1$ for $1 \leq i \leq P$, and an input DAG $G = (V, E, \omega, c \equiv 0)$. Given a bound C , we ask whether there exists a valid schedule whose total carbon cost does not exceed C . We prove in [32, Appendix A.3] that UCAS is strongly NP-complete by reducing the well-known 3-Partition problem to it. \square

4.3 Integer linear program

We formulate the problem as an integer linear program. Due to space constraints, we only sketch the derivation and refer to [32, Appendix A.4] for details. The ILP is written in terms of time units, hence it has a pseudo-polynomial number of variables. As stated in Section 3, the objective function is then to minimize

$$CC = \sum_{t=0}^{T-1} \max \left(\sum_{i=1}^{p^2} (\mathcal{P}_{\text{idle}}^i + \delta(t, i) \mathcal{P}_{\text{work}}^i) - \mathcal{G}_t, 0 \right), \quad (2)$$

where $\delta(t, i)$ is a boolean variable that specifies whether processor p_i is active at time t . Note that we still use the communication-enhanced graph, and the ILP enforces all dependence constraints and guarantees that all tasks are completed by the deadline T .

5 Algorithms

In this section, we present CaWoSched, a carbon-aware workflow scheduler for the scheduling problem of minimizing the carbon cost, given a mapping and a deadline. Recall that we work on the communication-enhanced DAG $G_c = (V_c, E_c, \omega)$. In Section 5.1, we first present the baseline algorithm, ASAP (As Soon As Possible), which schedules each task at its earliest possible start time, without taking the intervals into account. Section 5.2 presents several variants of a greedy procedure that allocates start times to tasks, building on a score that is computed for each task. Finally, we explain in Section 5.3 how to further improve the schedule obtained by the greedy algorithm, by using local search.

5.1 Baseline algorithm

The ASAP baseline algorithm starts each task at their earliest possible start time (EST). To compute these times, we proceed similarly to the computation of a topological ordering.

For all tasks $u \in V_c$ with in-degree 0 (guaranteed to exist since G_c is acyclic), we set $EST(u) = 0$ and decrease by one the in-degree of successor tasks (tasks v such that $(u, v) \in E_c$). A task v obtains an in-degree of 0 once all of its predecessors have been handled, and we can then compute its earliest start time as:

$$EST(v) = \max_{(u, v) \in E_c} \{EST(u) + \omega(u)\},$$

which corresponds to the time when all predecessors have completed their execution, when they are started as soon as possible. The computation of EST is done with a queue to handle tasks. The

proof for correctness and existence is similar to the proof of correctness for Kahn’s algorithm for topological sorting [21] and is hence omitted here.

5.2 Greedy schedule

We now describe how to compute a greedy schedule for the workflow, while accounting for the carbon cost of each interval (ASAP does not consider intervals at all). The idea is to assign a score to each task, and sort the tasks accordingly. Afterwards, we process the tasks in this order and try to find a good starting time for them.

Scores for the tasks. The goal of the scores is to express how beneficial it is to schedule a task before other tasks.

The first score is the **slack** $s(v)$ of task v , which represents the difference between the latest possible starting time of a task v , $LST(v)$, and its earliest start time $EST(v)$.

LST can be computed similarly to EST , using a queue to handle tasks. We set $LST(v) = T - \omega(v)$ if $v \in V_c$ and decrease by one the out-degree of predecessor tasks (tasks u such that $(u, v) \in E_c$). A task u obtains an out-degree of 0 once all of its successors have been handled, and we can then compute its latest start time as:

$$LST(u) = \min_{(u,v) \in E_c} \{LST(v) - \omega(u)\}.$$

Hence, the slack $s(v) = LST(v) - EST(v)$ describes the number of time units by which a task v can be shifted, since its start time has to be between $EST(v)$ and $LST(v)$. If the slack of a task is large, then it usually means that we have some flexibility to schedule it. We therefore try first to schedule tasks with a small slack, since there will still be room to shift tasks with a higher slack later. Note, however, that the slack does not account for the running time of the task.

The second score is the **pressure** of a task v , defined as:

$$\rho(v) = \frac{\omega(v)}{s(v) + \omega(v)}.$$

While slack does not take the running time into account, pressure accounts for it since it might play an important role for the power usage of the cluster. For pressure values, we have $0 \leq \rho(v) \leq 1$, with a pressure of 1 when there is no flexibility (i.e., $s(v) = 0$).

Hence, on the one hand, there is a high pressure to schedule a task v if its running time is large compared to the range in which it can run. On the other hand, if a task has low pressure, it means that there is a lot of flexibility for starting the task. In this case, it is beneficial to schedule tasks with high pressure first; hence, we sort the tasks by non-increasing order of pressure.

However, both scores do not account for the heterogeneity of the processors in terms of power consumption. Hence, we also introduce two weighted scores, where the functions for a task v mapped on processor p_i are multiplied by the factor:

$$wf(i) = \frac{\mathcal{P}_{\text{idle}}^i + \mathcal{P}_{\text{work}}^i}{\max_j (\mathcal{P}_{\text{idle}}^j + \mathcal{P}_{\text{work}}^j)}$$

for pressure and its reciprocal for slack. For slack, we use the reciprocal since tasks are sorted in non-decreasing order.

Subdivision of the intervals. Recall that I_1, \dots, I_J are the initial intervals coming from the power profile. As discussed in Section 4.1, there is a more fine-grained subdivision of these intervals, such that every task starts at the beginning of such an interval when we

look at the special case of one processor. Motivated by this result and the question of how to find a good starting time for a task without looking at every time unit, we do a similar subdivision for the multiprocessor case. On each processor, we create all possible blocks of at most $k = 3$ consecutive tasks (the parameter k is used to limit the number of intervals, and hence the time complexity of the heuristics). Each block is tentatively scheduled to start or end at one of the original intervals, and we memorize the possible

start times for each task and each block. When this is done on all processors, we sort the possible start times and compute the induced subdivision of the intervals.

Further refinement could be used by considering larger block sizes $k > 3$, but we observed in our experiments that $k = 3$ already creates a lot of subintervals.

Algorithm variants without local search. With four scores (slack, pressure, weighted-slack, weighted-pressure) and two interval subdivisions (normal or refined), we obtain eight algorithm variants: slack (unweighted, normal), slackW (weighted, normal), slackR (unweighted, refined), slackWR (weighted, refined) for slack and analogously with prefix press for pressure.

We now detail how these algorithms select a starting time for each task, which is always a time at the beginning of an interval.

Given a score and an interval subdivision, we pick the next task, say v , according to the best score value. The interval set is denoted as $\{I_1, \dots, I_{J'}\}$, where $J' = J$ if intervals are not refined, and $J' \geq J$ otherwise. We have $I_j = [b_j, e_j]$. First, the algorithm computes the subset of the intervals such that $EST(v) \leq b_j \leq LST(v)$, i.e., intervals at the beginning of which the task can be started.

If this set is empty (which is rarely the case in practice), we simply start the task at time $EST(v)$. Otherwise, we sort the intervals according to their budget \mathcal{G}_j and schedule the task to start at the beginning of the interval with the highest budget. If there are multiple intervals that are possible, we use the interval with the earliest starting point.

Afterwards, we look at all intervals during which task v runs. For the first and last intervals, if v does not cover the whole interval, we split the interval in two sub-intervals (one where the task is running, the other where it is not). Then, on each interval where the task runs, we decrease the power budget by $\mathcal{P}_{\text{idle}}^i + \mathcal{P}_{\text{work}}^i$, where p_i is the processor on which task v is mapped, to account for the fact that there is now a task running in this interval and consuming some power – hence the green budget is lower.

Also, once the task has been scheduled, this influences the EST and LST of other tasks as well. Hence, we update this for all tasks that have not been scheduled yet. In particular, these updates have to be made possibly for the whole graph, and we use a precomputed topological order for this. These updates take $O(n + |E_c|)$ time.

5.3 Local search

Once a greedy schedule has been obtained, we propose to refine this schedule by doing a local search, exploiting the flexibility that tasks still provide within the greedy schedule. The corresponding algorithm variants receive a suffix of -LS, for example pressWR-LS.

For the local search, we introduce a parameter $0 \leq \mu \leq T - 1$. First, we sort the processors by non-increasing power consumption $\mathcal{P}_{\text{work}}^i$, i.e., the more costly processor is considered first. For

each processor in this order, we then iterate over the tasks of the processors from left to right and look τ time units to the left and right, and check whether moving the task would give us a gain and is valid. This means that we make sure for every possible move that the corresponding start time of a task v stays in the interval $[EST(v), LST(v)]$. To this end, we iterate over the time units from the earliest to the latest. If we find a legal move with a positive gain, we apply it and update the cost. (One could also check all legal moves and apply the best one. However, preliminary experiments showed that this would not significantly improve the outcome, so we opted for the faster variant.) Once this has been done for all tasks on the current processor, we process the tasks of the next processor in the ranking. At each round, we record whether we had a positive gain or not. If there was one round through the tasks without gain, we stop the local search.

6 Experimental Evaluation

In this section, we evaluate the proposed carbon-aware scheduling framework CaWoSched with its numerous algorithm variants. We mainly focus on solution quality and running time in comparison to the baseline ASAP. For small instances, we also compare the quality against optimal solutions derived from the ILP formulation. The code and data used in the simulations are publicly available for reproducibility purposes at <https://github.com/KIT-EAE/CaWoSched>.

6.1 Simulation setup

Target computing platform. We consider two target computing platforms with a heterogeneous setup; their properties are inspired by real-world machines used for the experimental evaluation in [6]. There are six processor types, and we consider 12 (resp. 24) nodes of each type for the `small` (resp. `large`) cluster, hence a total of 72 (resp. 144) compute nodes.

In addition to the normalized speed values (see Table 1), we assign each processor a value for its idle power consumption $\mathcal{P}_{\text{idle}}$ and its active power consumption $\mathcal{P}_{\text{work}}$. The values for the power consumption are inspired by values coming from Intel [20] for modern processors. Note that we did not choose $\mathcal{P}_{\text{work}}$ too large since the CPU utilization in data centers is often far from 100% [1]. While the correlation between power consumption and processor speeds may not be obvious, the general trend is that faster processors consume more power, hence a ranking of the processor types: nodes of type `PT1` are the slowest/least consuming nodes, up to `PT6`, which are the fastest/most consuming ones. According to [1], the power consumption of the network is much smaller than that of computation, hence we draw the values for $\mathcal{P}_{\text{idle}}$ and $\mathcal{P}_{\text{work}}$ randomly between 1 and 2 for communication links, in order to introduce a small amount of heterogeneity.

Workflows and mappings. We evaluate the presented algorithmic framework on 34 workflows. The corresponding DAGs can be divided into real-world workflows obtained from [6] (`ataseq`, `bacass`, `eager` and `methyseq`) and workflows obtained by simulating real-world instances using the WGen generator [11]. We transformed the corresponding definition for the workflow management system Nextflow [12] to a `.dot` format with a Nextflow tool. Since the resulting DAGs contain many pseudo-tasks that are only internally relevant for Nextflow, we deleted them, following what was done

in [22]. For the simulated workflows, we use one of the respective real-world instances as a model graph and scale it up in size. As number of vertices, we use 200, 1000, 2000, 4000, 8000, 10000, 15000, 18000, 20000, 25000 and 30000. Every graph has vertex and edge weights following a normal distribution, where we make sure that the vertex weights are in general larger than the edge weights. Note that these are normalized values, and the actual running time of the task is determined by its vertex weight and its assigned processor. We normalize the network communication bandwidth to 1 since its influence is not considered here.

Furthermore, we generate for every graph two mappings, one for cluster `small` and one for cluster `large`. The mappings are generated with our own basic HEFT implementation without special techniques for tie-breaking, because that would not change the fact that HEFT is not carbon-aware. Since there are more fast and power-intensive processors on the `large` cluster, HEFT schedules more tasks to these processors and hence there are fewer tasks per processor on the other processors, compared to the `small` cluster.

Power profiles. For each workflow, we generate four differently shaped (renewable) energy profiles for different scenarios. We make sure that green power is always at least the sum of the idle power of the processors and at most the sum of idle power and 80% of the sum of the work power. The rationale is as follows: if we do not have enough green power or more green power than required overall, the decisions of the scheduler become irrelevant. Hence, we try to create scenarios where scheduling decisions have to be done in a smart way. The scenarios are the following:

- S1: A $-x^2$ shape, where the interval budgets follow this function with random perturbations. This models a situation where there is little green power in the beginning, then the supply with green energy is rising and falls at some point again (solar power from morning to evening, for example).
- S2: An x^2 shape that models the same situation as in S1, but starting from midday, again with random perturbations.
- S3: A $\sin(x)$ shape, where we model 24 hours of this scenario, i.e., we have little green power in the beginning and then we follow a sinus shape as given on $[0, 2\pi]$. We also add random perturbations.
- S4: A constant green power budget with perturbations (which can model situations where one has storage for renewable energy or nuclear power – see setting of France in [39]).

For each scenario, we have four different deadlines. Let D be the time required by the ASAP schedule, which is the tightest deadline. We consider deadlines D , $1.5D$, $2D$, and $3D$, providing more or less flexibility to shift tasks around in the schedule.

Table 1: Processor specifications in the clusters.

Processor Name	Speed	$\mathcal{P}_{\text{idle}}$	$\mathcal{P}_{\text{work}}$	small	large
<code>PT1</code>	4	40	10	×12	×24
<code>PT2</code>	6	60	30	×12	×24
<code>PT3</code>	8	80	40	×12	×24
<code>PT4</code>	12	120	50	×12	×24
<code>PT5</code>	16	150	70	×12	×24
<code>PT6</code>	32	200	100	×12	×24

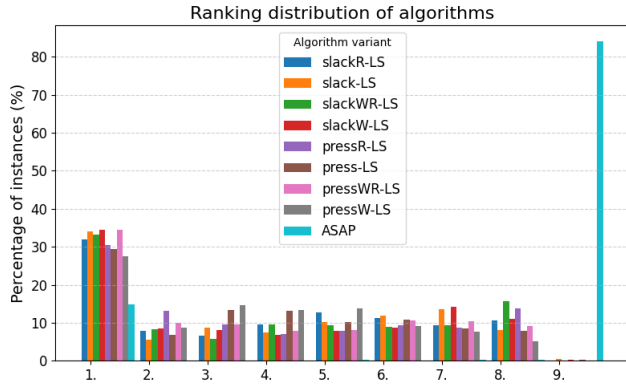


Figure 1: The distribution shows for which percentage of the instances each algorithm variant was ranked first, second, third, etc. Note that multiple algorithm variants can have the same rank.

Hence, we have in total 16 power profiles. For the workflow types *atacseq* and *methyleseq*, we have 12 graphs per type, for *bacass* we only use the real-world version due to problems with scaling, and for *eager* we have 9 graphs with up to 18000 vertices. This results in $2 \times 34 \times 16 = 1088$ simulations (2 platforms, 34 workflows, 16 power profiles) per algorithm. All algorithms are implemented in C++ and compiled with g++ (v.13.2.0) with compiler flag `-O3`. The experiments are managed by *simexpal* [4] and executed on workstations with 192 GB RAM and 2x 12-Core Intel Xeon 6126 @3.2 GHz and CentOS 8 as OS. The ILP is implemented using Gurobi’s Python API [16] and for license reasons executed on a machine with a 13th Gen Intel(R) Core(TM) i7-1355U processor with 16GB RAM running Ubuntu 24.04.1 LTS. Further, for the simulation results below, we set the tuning parameter for the subdivision to $k = 3$ and the tuning parameter for the local search to $\mu = 10$.

6.2 Simulation results

We compare the quality of the schedules returned by the *CaWoSched* variants. Recall that there are two base scores, *slack* and *pressure*, that can be weighted by a factor accounting for the heterogeneity in power consumption of the processors, and we can use either the original or refined intervals (see Section 5). The heuristics then apply a local search to further improve the solution. We first compare the solution quality when the local search is applied, but we also analyze the influence of the local search on different algorithm variants. Next, we study the impact of various parameters. We also compare the heuristics’ solution to the optimal solution returned by the ILP on small instances.

With local search. As a first measure of performance, we rank the different algorithm variants for each instance. This means that we record the frequency with which each algorithm variant ranks first, second, third, etc., in terms of carbon cost. Note that this means that if two variants have the same cost, they will end up with the same rank and the next rank is then skipped. The results can be seen in Figure 1; the main observations are the following. First, we can see that all our algorithm variants are ranked first significantly more often than this is the case for the baseline *ASAP*. Note that even if

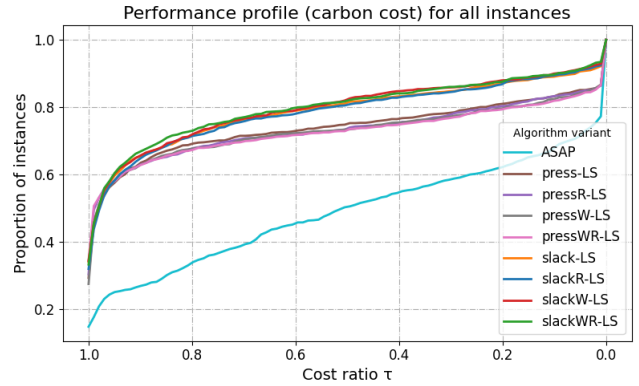


Figure 2: The ratio is the best cost found divided by the algorithm variants’ own cost. Then the percentage of instances for which this is larger than or equal to τ is plotted. A higher curve is better.

the baseline has rank 1, it does not necessarily mean that a variant of the algorithm performed worse, since they all could have found the optimal solution. In particular, we can see that the baseline performed worst in 84.01% of the cases. Another observation is that none of the algorithm variants does significantly outperform the others in terms of ranking. The *pressWR-LS* variant is ranked first most frequently (34.47%), but with a small margin.

If we look at performance profiles, we obtain more detailed insights about the quality of each variant, see Figure 2. We report the proportion of instances with a cost ratio at most 1, where the cost ratio is the best carbon cost divided by the algorithm’s carbon cost. Note that if the algorithm’s carbon cost is 0, then the best cost is also 0 and the ratio is set to 1. Otherwise, a ratio of 1/2 means that the heuristic’s cost is twice higher than the best cost, and a ratio of 0 corresponds to a non-null carbon cost, while the best is 0. Even though we can see here again that for $\tau = 1.0$, i.e., the proportion of instances for which the algorithm variant achieves the best cost, is the highest for *pressWR-LS*, we also observe that for lower τ values, i.e. more suboptimality tolerance, the curves for the algorithm variants using *slack* as base score surpass the *pressure* variant, which hints at a better overall performance on average. Interestingly, this observation seems to be influenced by the tolerance in the deadline. This is why we next illustrate how the deadline impacts the performance profile, as shown in Figure 3. While we can observe for a tight deadline that *pressR* and *pressWR* have a higher curve, these variants are clearly surpassed by *slack* variants when there is more tolerance in the deadline.

Another important aspect for the evaluation is the cost improvement over the baseline algorithm *ASAP*. For this, we first look at the median of the cost ratio between the baseline *ASAP* and the different algorithm variants over all instances. This is shown in Figure 4. (Note that a geometric mean is not applicable here because the ratio can be 0 if our heuristic has carbon cost zero but the baseline has not. Further, since there are cases when the baseline performs better than our heuristics, we cannot use the arithmetic mean, either.) We can see that all algorithms are closely together with a cost ratio median of ≈ 0.6 , meaning that the algorithm needs only 60% of the carbon cost compared to the baseline (or, vice versa,

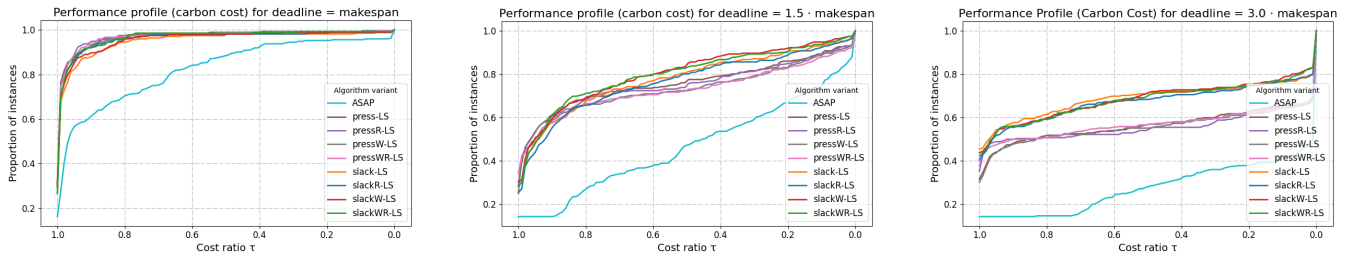


Figure 3: The evolution of the performance profiles when adding more tolerance to the deadline from left to right (data for deadline factor 2.0 can be found in [32, Appendix A.5]).

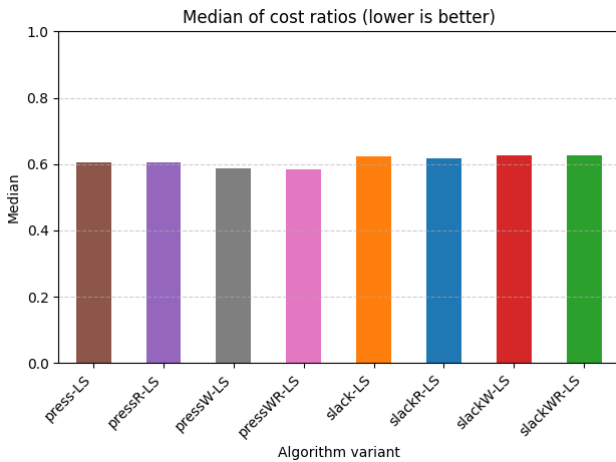


Figure 4: The median of the cost ratios obtained by dividing heuristics carbon cost by the carbon cost of the deadline.

they are $\approx 1.67\times$ better). We also see that regarding this cost ratio, the algorithm versions using pressure as base score perform better than the slack variants – pressWR-LS has the best cost ratio median with 0.58. Again, we observe the impact of exploiting more flexibility in terms of deadline in Figure 5, where it becomes clear that the cost ratio improves with more flexibility. There, we can see that while the gains for a tight deadline are moderate, the algorithm slackW has a cost ratio of only 0.15 compared to the baseline (or, vice versa, $\approx 6.67\times$ better). This is a behavior that we expected from the algorithms, since we have more opportunities for scheduling the tasks with an increased deadline.

To further investigate the improvement over the baseline, we look at boxplots for the improvement. The results for all instances are shown in Figure 6. What we can see here is that the solutions for most instances lie in between ≈ 0.25 and ≈ 0.9 , with most medians around 0.6 (compared to the baseline). We also see that, for some instances, the baseline performs better than the proposed algorithms. One reason for this is that some power profiles provide a lot of green power in the beginning of the horizon $[0, T]$. Hence, for these profiles, scheduling the tasks as soon as possible might be the best strategy. However, overall one can observe that this is rarely the case and that the new algorithms significantly improve over ASAP in terms of carbon cost.

Influence of local search. While we have studied so far the heuristics’ behavior when the local search was applied, we also run four

Table 2: Minimum, maximum and average cost ratio for comparing the algorithm with and without local search.

Algorithm Variant	Min	Max	Avg
slackR	0	1.0	0.25
slackWR	0	1.0	0.25
pressR	0	1.0	0.25
pressWR	0	1.0	0.23

of the heuristics without the local search to assess how much gain, in terms of carbon cost, can be achieved thanks to the local search. Note that we use a subset of the full test set for this approach, namely all variants of the atacseq workflow type and the bacass workflow. However, note that this still results in more than 400 experiments per algorithm variant. We report the minimum, maximum and average improvement in Table 2. Note that here we use the arithmetic mean since the geometric mean is not applicable because we can have a cost ratio of 0. (The results are still meaningful since we only have values between 0 and 1.) We can see that for every variant the cost ratio ranges from 0 to 1.0. A cost ratio of 0 comes from instances where the algorithm achieves zero carbon cost using local search, but the algorithm variant without local search has positive carbon cost. Note that a cost ratio larger than 1.0 is not possible since the local search approach is designed as a hill climber. In general, we can see that our local search approach significantly improves the solution of the initial schedule. In particular, we can observe in our experiments that there is a significant number of instances where the local search approach reaches an optimal solution of 0, but the initial schedule has still positive carbon cost. Further, we can see that the local search improves all algorithm variants by a similar margin.

Impact of parameters. A complete study highlighting the impact of all parameters through detailed results is available in [32, Appendix A.5]. First, as expected, the ASAP baseline performs better when there is a lot of green power at the beginning of the time horizon or when there is no huge change as in Scenarios S4 or S2. We provide detailed results for each power profile, while we have presented so far aggregated results. Further, we can see that our algorithm achieves a significantly better cost ratio when there is not much green power in the beginning such as in Scenarios S1 and S3.

Also, we look separately at the cost ratios depending on the cluster size. We can see here that the cluster size has no significant influence on the performance of our heuristics. However, it

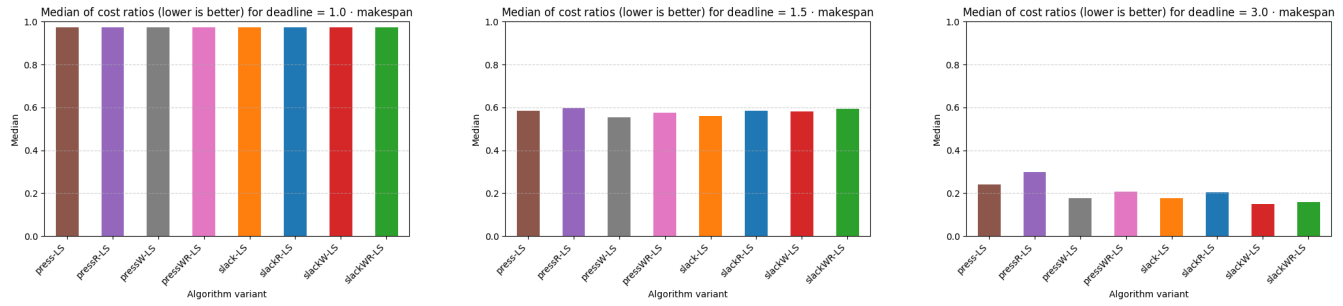


Figure 5: The evolution of the median of the cost ratios when adding more tolerance to the deadline from left to right (data for deadline factor 2.0 can be found in [32, Appendix A.5]).

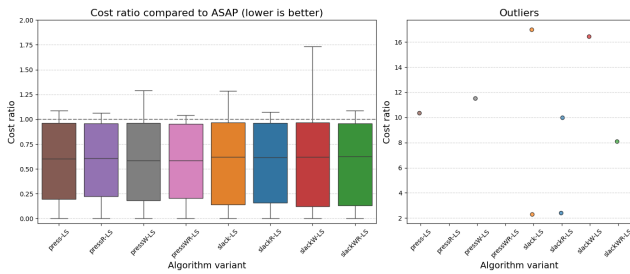


Figure 6: Boxplot of the cost ratios obtained by dividing the heuristics carbon cost by the carbon cost of the baseline. Outliers are shown in the separate plot on the right.

influences the performance profile. While for the large cluster, the curves are closer together, we see a similar situation as in Figure 2 for the smaller cluster.

Finally, we also study the impact of the number of tasks on the solution. The general trend is that the cost ratio gets slightly worse when the number of tasks increases. However, the effect is not significant, and we can conclude that the improvement of our heuristic over the baseline is in all cases significant.

Comparison with optimal solutions. We explore the quality of the novel heuristics when compared to an exact solution. For this, we use the ILP formulation presented in Section 4.3, and use the ILP solver Gurobi [16]. Further, our implementation in Python uses the NetworkX library [17]. We restrict to instances with up to 200 tasks, since the solver takes too long on larger instances (already up to one hour for 200 tasks vs only milliseconds for each heuristic). Note that the scope of this work is not to explore an efficient ILP formulation for this algorithm, we solely want to explore the quality of the novel algorithms. This is why we keep a simple but correct ILP with time units instead of moving to an interval formulation.

We show an analysis of the results in Figure 7. What we can see here is that the median cost ratio is still reasonable when we compare our heuristics to exact solutions. Further, this seems to be an achievement of our heuristics since we can see that the baseline has a much worse cost ratio than our heuristics. Further, it is interesting to see that there are a significant number of instances where the cost ratio is 1.0, meaning that our heuristic is able to achieve the optimal solution.

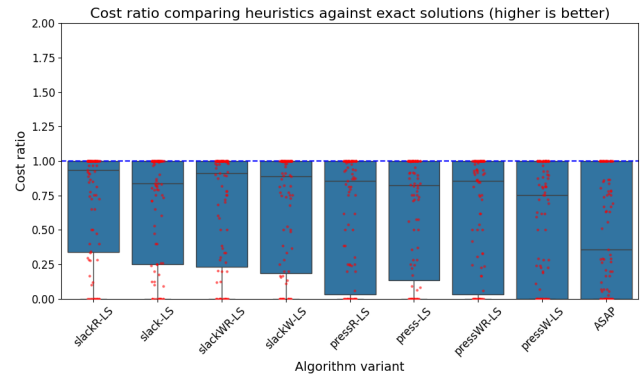


Figure 7: Cost ratio obtained by dividing the ILP result through the heuristic result. Red dots show the actual cost ratios.

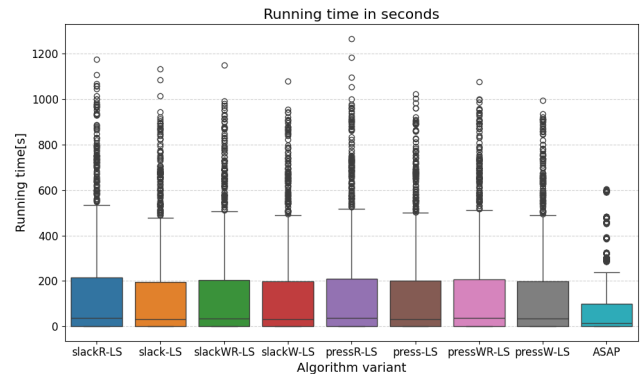


Figure 8: Time (in seconds) for each algorithm variant.

6.3 Running Time Evaluation

In this section, we present the time needed by the various algorithm variants for computing a carbon-aware schedule. Even though it is not the main goal of the scheduler to be as fast as possible, it is important that it is not overly time-consuming to be applicable in real-world scenarios.

Aggregated running time values based on all workflows are shown in Figure 8. We can observe that all algorithm variants yield a reasonable slowdown compared to the baseline. For most of the instances, the scheduler is able to compute a schedule within

seconds, while larger workflows with up to 30000 tasks can take several minutes.

In [32, Appendix A.5] we can see that the largest running times result from large workflows with 20000 to 30000 tasks. Overall, most of the instances are still solvable in less than a few minutes.

Another interesting aspect is that the running time seems to be mainly influenced by graph size, but not by the length of the time horizon T . The running time increases only slightly with an increased deadline, which indicates that the algorithms successfully make decisions based on structural graph information – without having a too broad search tree for each task if the deadline increases. The data leading to this insight can be found in [32, Appendix A.5].

7 Conclusions

This work aimed at minimizing carbon emissions when executing a scientific workflow on a parallel platform with a time-varying mixed (renewable and non-renewable) energy supply. We focused on improving a given mapping and ordering of the tasks (for instance generated by HEFT) by shifting task executions to greener time intervals whenever possible, while still enforcing all dependencies. We showed that this algorithmic problem can be solved in polynomial time in the uniprocessor case. For two processors, the problem becomes NP-hard, even for a simple instance with independent tasks and carbon-homogeneous processors. We proposed a heuristic framework combining several greedy approaches with local search. The experimental results showed that our heuristics provide significant savings in carbon emissions compared to the baseline. Furthermore, for smaller problem instances, we showed that several heuristics achieve a performance close to the optimal ILP solution. Altogether, all these results represent a major advance in the understanding of the problem.

Future work will be devoted to the next step, namely targeting the design of a carbon-aware extension of HEFT. Mapping and scheduling the workflow at the same time while minimizing carbon emissions may well lead to even better solutions. Given the difficulty of the problem, we envision a two-pass approach: a first pass devoted to mapping and ordering, but without a finalized schedule, and a second pass devoted to optimizing the schedule through the approach followed in this paper.

Acknowledgements. This work is partially supported by Collaborative Research Center (CRC) 1404 FONDA – *Foundations of Workflows for Large-Scale Scientific Data Analysis*, which is funded by German Research Foundation (DFG).

References

- [1] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. 2010. Energy proportional datacenter networks. In *Proc. of the 37th Annual Int. Symp. on Computer Architecture*. 338–347. doi:10.1145/1815961.1816004
- [2] Mainak Adhikari, Tarachand Amgoth, and Satish Narayana Srirama. 2019. A Survey on Scheduling Strategies for Workflows in Cloud Environment and Emerging Trends. *ACM Comput. Surv.* 52, 4 (2019). doi:10.1145/3325097
- [3] Kazi Main Uddin Ahmed, Math H. J. Bollen, and Manuel Alvarez. 2021. A Review of Data Centers Energy Consumption and Reliability Modeling. *IEEE Access* 9 (2021), 152536–152563.
- [4] Eugenio Angriman, Alexander van der Grinten, Moritz von Looz, Henning Meyerhenke, Martin Nöllenburg, Maria Predari, and Charilaos Tzovas. 2019. Guidelines for Experimental Algorithmics: A Case Study in Network Analysis. *Algorithms* 12, 7 (2019), 127. doi:10.3390/A12070127
- [5] Hamid Arabnejad and Jorge G Barbosa. 2013. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. on Par. and Distr. Systems* 25, 3 (2013), 682–694.
- [6] Jonathan Bader, Fabian Lehmann, Lauritz Thamsen, Ulf Leser, and Odej Kao. 2024. Lotaru: Locally predicting workflow task runtimes for resource management on heterogeneous infrastructures. *Future Generation Computer Systems* 150 (2024).
- [7] Jonathan Bader, Kathleen West, Soeren Becker, Svetlana Kulagina, Fabian Lehmann, Lauritz Thamsen, Henning Meyerhenke, and Odej Kao. 2025. Predicting the Performance of Scientific Workflow Tasks for Cluster Resource Management: An Overview of the State of the Art. arXiv:2504.20867 [cs.DC]
- [8] Jorge G Barbosa and Belmiro Moreira. 2011. Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters. *Parallel computing* 37, 8 (2011).
- [9] Enno Breukelman, Sophie Hall, Giuseppe Belgioioso, and Florian Dörfler. 2024. Carbon-Aware Computing in a Network of Data Centers: A Hierarchical Game-Theoretic Approach. In *2024 Europ. Control Conf. (ECC)*. IEEE, 798–803.
- [10] Zhiwei Cao, Xin Zhou, Han Hu, Zhi Wang, and Yonggang Wen. 2022. Toward a systematic survey for carbon neutral data centers. *IEEE Comm. Surveys & Tutorials* 24, 2 (2022), 895–936.
- [11] Tainã Coleman, Henri Casanova, Loïc Pottier, Manav Kaushik, Ewa Deelman, and Rafael Ferreira da Silva. 2022. WfCommons: A framework for enabling scientific workflow research and development. *Future Generation Computer Systems* 128 (2022), 16–27. doi:10.1016/j.future.2021.09.043
- [12] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature biotechnology* 35, 4 (2017), 316–319.
- [13] Juan J. Durillo, Vlad Nae, and Radu Prodan. 2014. Multi-Objective Energy-Efficient Workflow Scheduling Using List-Based Heuristics. *Future Generation Computer Systems* 36 (July 2014), 221–236. doi:10.1016/j.future.2013.07.005
- [14] Juan J. Durillo, Radu Prodan, and Jorge G. Barbosa. 2015. Pareto Tradeoff Scheduling of Workflows on Federated Commercial Clouds. *Simulation Modelling Practice and Theory* 58 (Nov. 2015), 95–111. doi:10.1016/j.simpat.2015.07.001
- [15] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company.
- [16] Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual.
- [17] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proc. of the 7th Python in Science Conf.*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). 11 – 15.
- [18] Sophie Hall, Francesco Micheli, Giuseppe Belgioioso, Ana Radovanović, and Florian Dörfler. 2024. Carbon-Aware Computing for Data Centers with Probabilistic Performance Guarantees. arXiv preprint arXiv:2410.21510 (2024).
- [19] Walid A. Hanafy, Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. 2023. CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 3 (2023).
- [20] Intel Corporation. 2011. *Measuring Processor Power - TDP vs. ACP*. White Paper. Intel Corporation. <https://www.intel.com/content/dam/doc/white-paper/resources-xeon-measuring-processor-power-paper.pdf>
- [21] A. B. Kahn. 1962. Topological sorting of large networks. *Commun. ACM* 5, 11 (Nov. 1962), 558–562. doi:10.1145/368996.369025
- [22] Svetlana Kulagina, Anne Benoit, and Henning Meyerhenke. 2025. Memory-aware Adaptive Scheduling of Scientific Workflows on Heterogeneous Architectures. arXiv:2503.22365 [cs.DC]
- [23] Svetlana Kulagina, Henning Meyerhenke, and Anne Benoit. 2024. Mapping Large Memory-constrained Workflows onto Heterogeneous Platforms. In *53rd Int. Conf. on Par. Processing (ICPP)*.
- [24] Hessam Lavi. 2023. Measuring greenhouse gas emissions in data centres: the environmental impact of cloud computing. <https://www.climatiq.io/blog/measure-greenhouse-gas-emissions-carbon-data-centres-cloud-computing>.
- [25] Ji Liu, Esther Pacitti, and Patrick Valduriez. 2018. A survey of scheduling frameworks in big data systems. *Int. J. of Cloud Computing* 7, 2 (2018), 103–128.
- [26] A. Hasan Mahmud and S. S. Iyengar. 2016. A Distributed Framework for Carbon and Cost Aware Geographical Job Scheduling in a Hybrid Data Center Infrastructure. In *IEEE Int. Conf. on Autonomic Comp. (ICAC)*. 75–84.
- [27] M. Yusuf Özkaya, Anne Benoit, Bora Uçar, Julien Herrmann, and Ümit V. Çatalyürek. 2019. A scalable clustering-based task scheduler for homogeneous processors using DAG partitioning. In *33rd IEEE Int. Par. and Distr. Proc. Symp.*
- [28] Paul Pop, Kåre Harbo Poulsen, Viacheslav Izosimov, and Petru Eles. 2007. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proc. of the 5th IEEE/ACM Int. Conf. on Hardware/Software Codesign and System Synthesis (Salzburg, Austria)*. 233–238.
- [29] Ana Radovanović, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, et al. 2022. Carbon-aware computing for datacenters. *IEEE Trans. on Power Sys.* 38, 2 (2022).
- [30] Yassir Samadi, Mostapha Zbakh, and Claude Tadonki. 2018. E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm for Task Scheduling based on Load Balancing in Cloud Computing. In *2018 Int. Conf. on High Performance Computing & Simulation (HPCS)*. 601–609. doi:10.1109/HPCS.2018.00100
- [31] Suhelash Sandokji and Fathy Eassa. 2019. Dynamic Variant Rank HEFT Task Scheduling Algorithm Toward Exascale Computing. *Procedia Computer Science* 163 (2019).

- [32] Dominik Schweisgut, Anne Benoit, Yves Robert, and Henning Meyerhenke. 2025. Carbon-Aware Workflow Scheduling with Fixed Mapping and Deadline Constraint. (2025). arXiv:2507.08725 [cs.DC] <https://arxiv.org/abs/2507.08725>.
- [33] Zhiao Shi and Jack J. Dongarra. 2006. Scheduling workflow applications on processors with different capabilities. *Future Gen. Comp. Sys.* 22, 6 (2006).
- [34] Oliver Sinnen. 2007. *Task scheduling for parallel systems*. Vol. 60. John Wiley & Sons.
- [35] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Par. and Distr. Systems* 13, 3 (2002), 260–274.
- [36] Laurens Versluis and Alexandru Iosup. 2022. Taskflow: An energy- and makespan-aware task placement policy for workflow scheduling through delay management. In *Proc. of the 2022 ACM/SPEC Int. Conf. on Perf. Eng.* 81–88.
- [37] Jaagup Viil and Satish Narayana Srirama. 2018. Framework for automated partitioning and execution of scientific workflows in the cloud. *The J. of Supercomputing* 74 (2018), 2656–2683.
- [38] Zhenyu Wen, Saurabh Garg, Gagangeet Singh Aujla, Khaled Alwasel, Deepak Puthal, Schahram Dustdar, Albert Y. Zomaya, and Rajiv Ranjan. 2021. Running Industrial Workflow Applications in a Software-Defined Multicloud Environment Using Green Energy Aware Scheduling Algorithm. *IEEE Trans. on Ind. Inf.* 17, 8 (2021).
- [39] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. 2021. Let’s Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud. In *Proc. of the 22nd Int. Middleware Conference*.