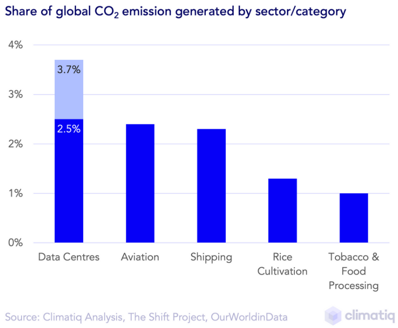
Carbon-Aware Workflow Scheduling with Fixed Mapping and Deadline Constraint

Dominik Schweisgut (Humboldt-Universität zu Berlin, Germany)
Henning Meyerhenke (KIT, Germany)
Anne Benoit, Yves Robert (ENS Lyon & IUF, France)

18th **Scheduling for large-scale systems** workshop
Montréal, Canada, July 8-10, 2025

Motivation

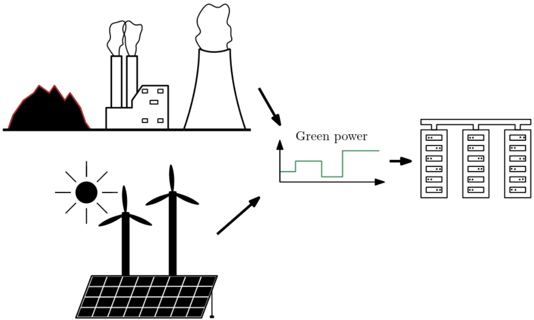
- Today's data centers generate more CO_2 than the aviation industry



- Reducing CO_2 emissions of data centers is of financial and environmental interest

Motivation

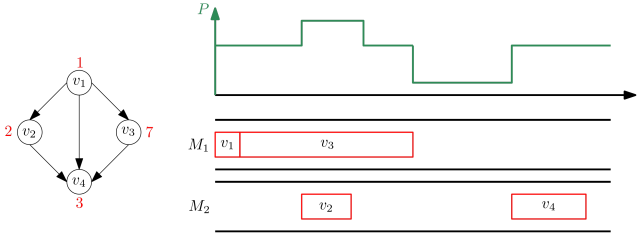
- The amount of cleaner power (solar, wind, nuclear) varies over the day



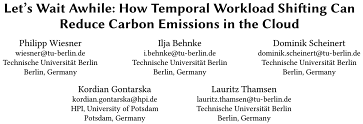
- Taking the data center's energy mix into account can help reduce CO_2 emissions from data centers

Approach

- **Question:** When computing schedules for a given workflow, can we find a way to exploit the variability of the energy mix?
- Prior research showed a large potential for time shifting of tasks in order to decrease carbon emissions [Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud, Wiesner et al, 2021]



Carbon-aware schedule



Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud

Philipp Wiesner, wiesner@tu-berlin.de, Technische Universität Berlin, Berlin, Germany
Ilja Behnke, i.behnke@tu-berlin.de, Technische Universität Berlin, Berlin, Germany
Dominik Scheinert, dominik.scheinert@tu-berlin.de, Technische Universität Berlin, Berlin, Germany
Kordian Gontarska, kordian.gontarska@hpi.de, HPI, University of Potsdam, Potsdam, Germany
Lauritz Thamsen, lauritz.thamsen@tu-berlin.de, Technische Universität Berlin, Berlin, Germany

State of the art

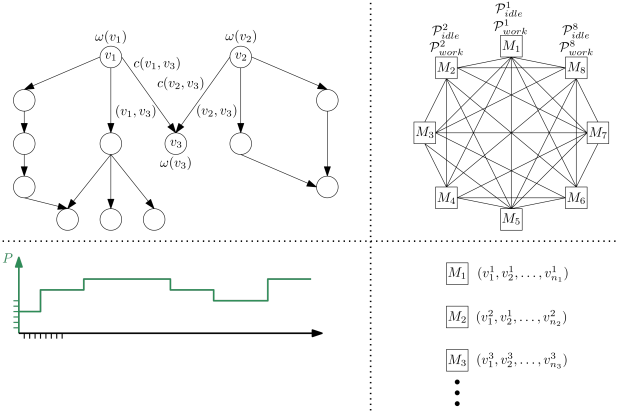
- Most schedulers today take scheduling decisions to minimize makespan, i.e., total execution time
- Recently,
  - Research with respect to energy consumption gained more attention due to energy cost and environmental concerns
  - Carbon emission reduction gained more attention, since it is and will become even more important regarding cost and environment

Contributions

- Focus on carbon footprint minimization in a setting where mapping and ordering of tasks and communications is fixed, with a constraint on the deadline, when it is possible to shift tasks
- Model and complexity study of this problem
  - Sophisticated fully polynomial-time DP algorithm with one processor
  - NP-completeness with at least two processors, even with independent tasks and homogeneous platform
  - ILP formulation of the general problem
- Design of efficient heuristics (`CaWoSched`) combining various greedy approaches with local search
- Extensive set of simulations to evaluate the achieved gains, in terms of carbon emissions

Outline

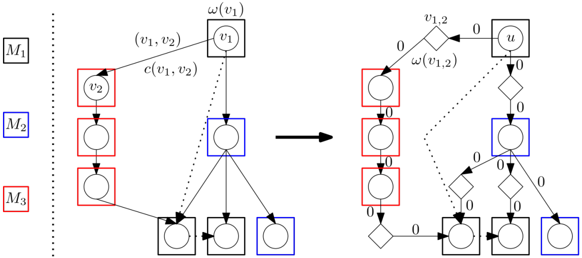
Model



Input DAG, cluster topology, power profile and mapping

- DAG with given mapping and fixed order of tasks/communications
- Full-duplex communication topology
- Idle processor power and dynamic power when computing, idem for communication links
- Power profile with amount of "green" power available with time

Communication-enhanced DAG

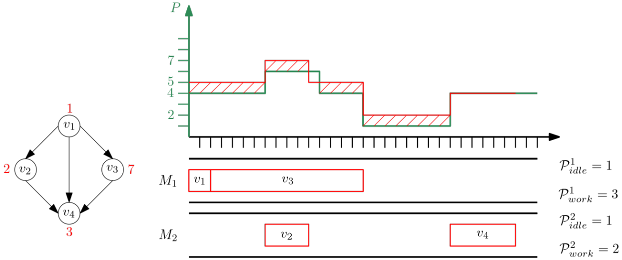


Communication-enhanced DAG G_c

- No communication if two tasks are on the same processor, but precedence constraint to express the ordering
- For inter-processor communications:
  - replace edge by path with communication task $v_{i,j}$
  - now $\omega(v_{i,j}) = c(v_i, v_j)$

$\Rightarrow$ edges represent only precedence constraints, additional processors for each communication channel

Problem statement



Example for brown excess power

- Sum up the power consumption of every processor per time unit
- Amount of 'brown power' per time unit is the excess power over the green power budget (also possible in polynomial time)
- We pay carbon cost per unit of brown power

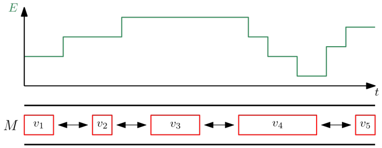
Objective: Find a schedule that minimizes carbon cost

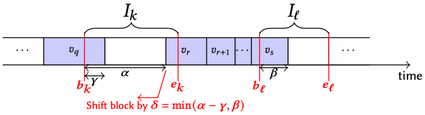
Outline

Single processor, n ordered tasks

Theorem

The problem instance with a single processor has polynomial-time complexity.



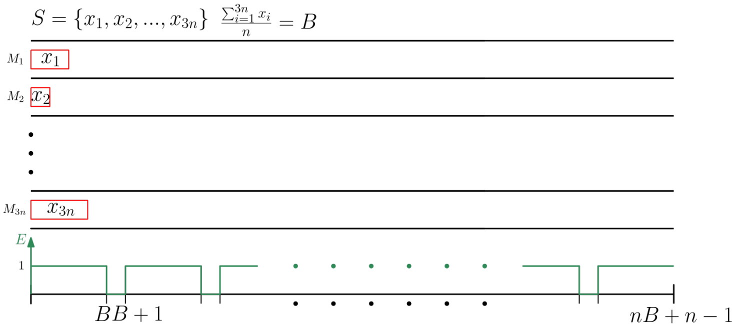


- **Idea:** There is an optimal schedule where consecutive blocks of tasks start or end at an interval start or end point
- Refined set of intervals that remains polynomial
- Dynamic programming algorithm considering all possible start times within this new set of intervals

Multiple processors

Theorem

The problem instance with several processors is strongly NP-complete, even with homogeneous processors and independent tasks.



- **Idea:** Reduction from 3-Partition – one task per processor, n intervals of size B with green power available for one processor

ILP formulation

- Pseudo-polynomial number of variables, considering all time units
- Boolean variable $\delta(t,i)$ to express whether processor p_i is active during time unit t
- Various constraints to enforce that the schedule is correct and that all tasks are completed before the deadline
- Objective function: minimize

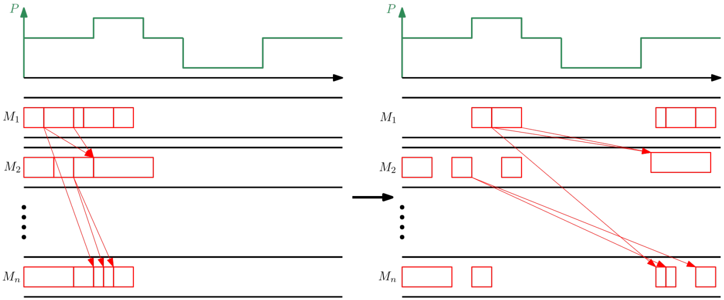
$$\mathcal{CC} = \sum_{t=0}^{T-1} \max\left(\sum_{i=1}^{P^2} \left(\mathcal{P}^i_{\text{idle}} + \delta(t,i)\mathcal{P}^i_{\text{work}}\right) - \mathcal{G}_t, 0 \right)$$

Outline

Algorithms for multi-processor case

Algorithms consist of **3** phases, using the communication-enhanced DAG:

1. Assign scores to tasks based on the deadline and the green power budget to determine the processing order of tasks
2. Greedily try to find optimal start times regarding carbon cost
3. Improve the obtained solution using local search



Scores

- Earliest start time of a task:

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

or 0

- Latest start time of a task:

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

or $T - \omega(v)$

- Task v has to start between $EST(v)$ and $LST(v)$
- **Baseline:** Start as soon as possible, i.e. at $EST(v)$

We have the following base scores:

- **(Weighted) Slack:** (ascending)

$$s(v) := (LST(v) - EST(v)) \cdot \frac{1}{wf(i)}$$

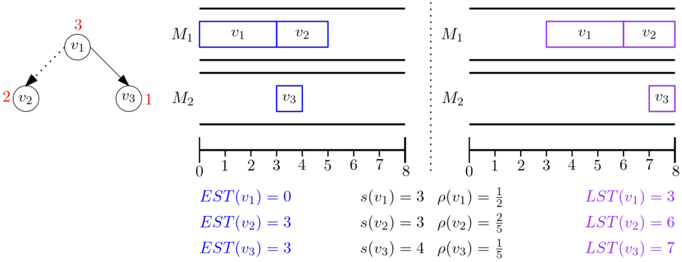
- **(Weighted) Pressure:** (descending)

$$\rho(v) := \left(\frac{\omega(v)}{s(v) + \omega(v)}\right) \cdot wf(i)$$

Optional **weight factor** if $proc(v) = i$:

$$wf(i) := \frac{\mathcal{P}^i_{\text{idle}} + \mathcal{P}^i_{\text{work}}}{\max_j\{\mathcal{P}^j_{\text{idle}} + \mathcal{P}^j_{\text{work}}\}}$$

Scores

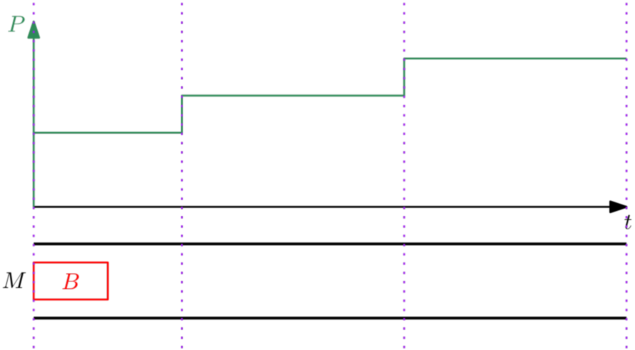


Difference between slack and pressure

- Slack makes no difference between tasks v_1 and v_2
- Pressure accounts for the larger running time of v_1 and assigns a higher value to v_1

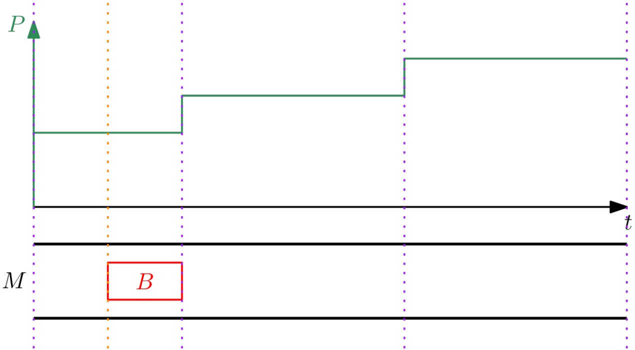
Greedy algorithm

- Compute a refined subdivision of the intervals, by tentatively placing blocks at appropriate start/end times



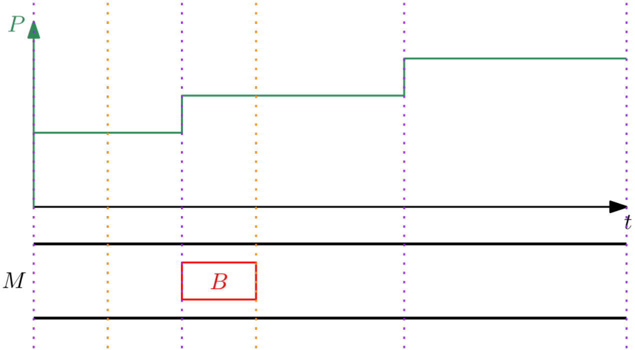
Greedy algorithm

- Compute a refined subdivision of the intervals, by tentatively placing blocks at appropriate start/end times



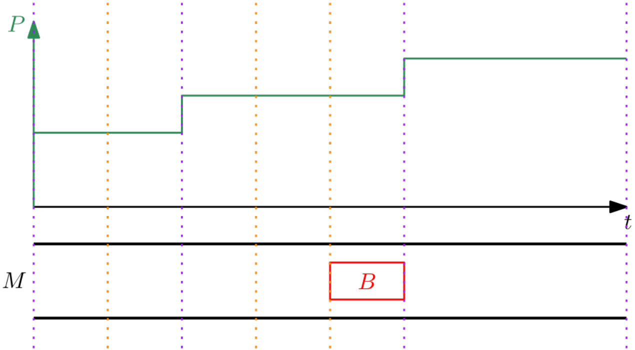
Greedy algorithm

- Compute a refined subdivision of the intervals, by tentatively placing blocks at appropriate start/end times



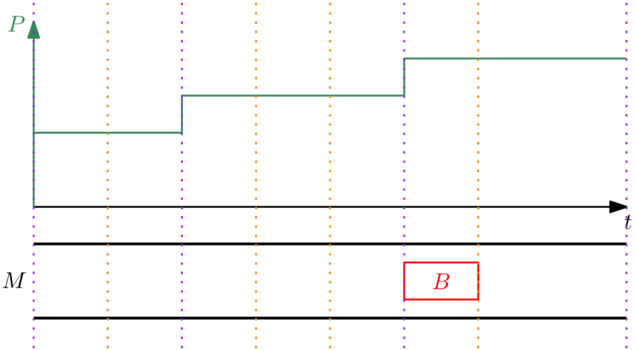
Greedy algorithm

- Compute a refined subdivision of the intervals, by tentatively placing blocks at appropriate start/end times



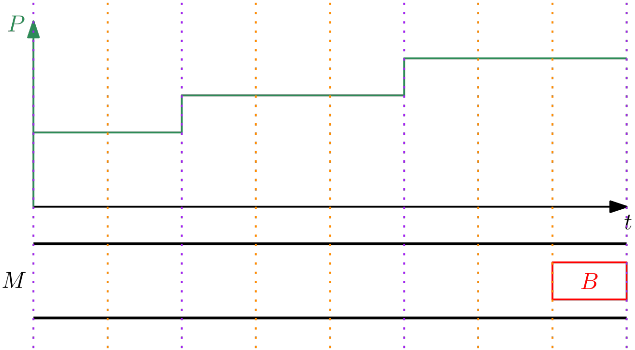
Greedy algorithm

- Compute a refined subdivision of the intervals, by tentatively placing blocks at appropriate start/end times

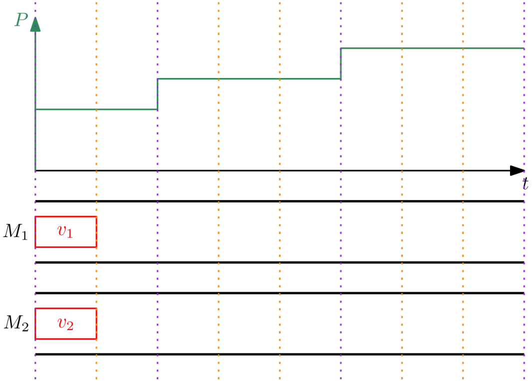


Greedy algorithm

- Compute a refined subdivision of the intervals, by tentatively placing blocks at appropriate start/end times

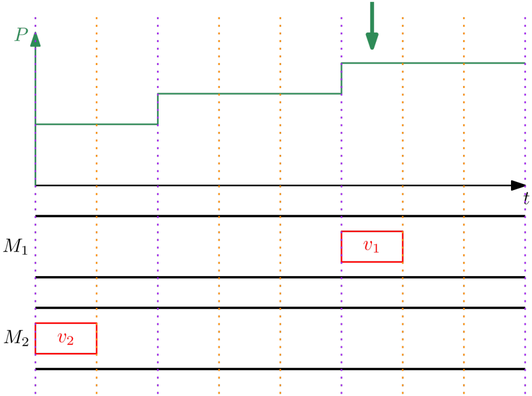


Greedy algorithm



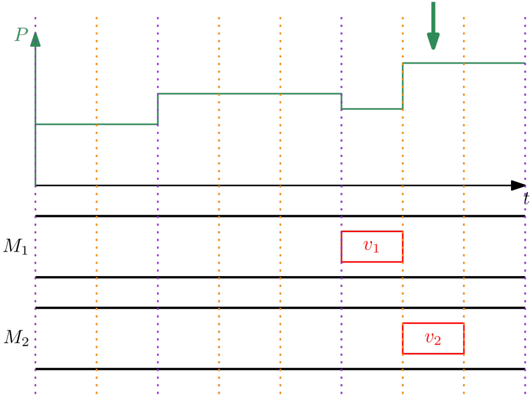
- Maintain power budget per (new) interval and greedily assign tasks to subintervals
- Split intervals if necessary
- Afterwards, update EST and LST for all dependent tasks, and hence corresponding score

Greedy algorithm



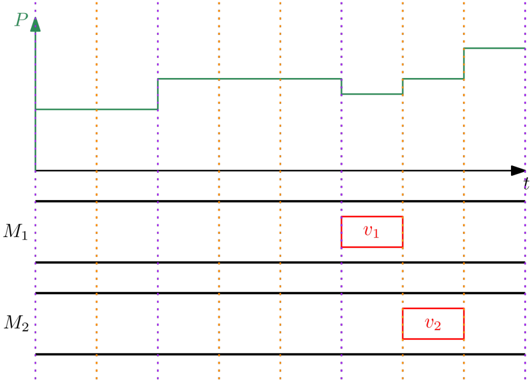
- Maintain power budget per (new) interval and greedily assign tasks to subintervals
- Split intervals if necessary
- Afterwards, update EST and LST for all dependent tasks, and hence corresponding score

Greedy algorithm



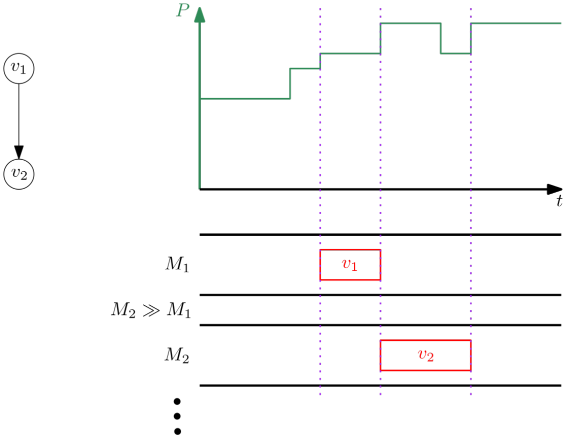
- Maintain power budget per (new) interval and greedily assign tasks to subintervals
- Split intervals if necessary
- Afterwards, update EST and LST for all dependent tasks, and hence corresponding score

Greedy algorithm



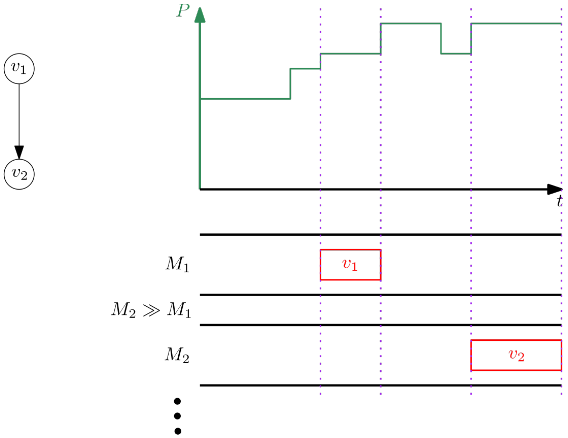
- Maintain power budget per (new) interval and greedily assign tasks to subintervals
- Split intervals if necessary
- Afterwards, update EST and LST for all dependent tasks, and hence corresponding score

Local search



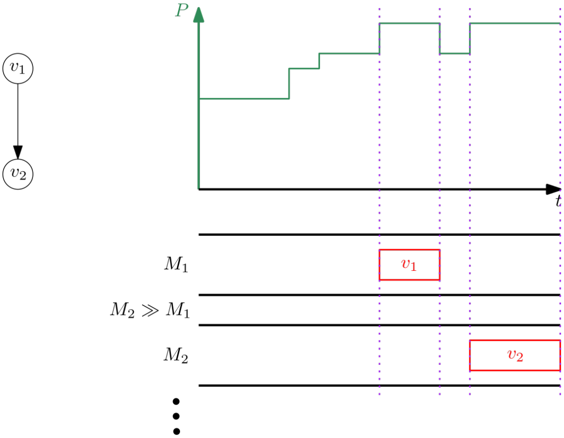
- **Idea:** Exploit the remaining flexibility in the schedule
- Sort processors with respect to their active power consumption $\mathcal{P}_{\text{work}}$
- Find among surrounding time units the best move that does not affect others for each task
- Hill climber

Local search



- **Idea:** Exploit the remaining flexibility in the schedule
- Sort processors with respect to their active power consumption $\mathcal{P}_{\text{work}}$
- Find among surrounding time units the best move that does not affect others for each task
- Hill climber

Local search



- **Idea:** Exploit the remaining flexibility in the schedule
- Sort processors with respect to their active power consumption $\mathcal{P}_{\text{work}}$
- Find among surrounding time units the best move that does not affect others for each task
- Hill climber

Outline

Setup

- Different cluster sizes: `small` (72 nodes) and `large` (144 nodes)
- 34 real-world and synthetic workflows with $200 - 30\,000$ tasks
- Mapping generated with HEFT [Topcuoglu et al., IEEE TPDS, 2002], and baseline **ASAP** using EST, finishing in time D
- Different deadlines: D (tight deadline), $1.5D$, $2D$ and $3D$

Different shapes for power profiles:

S1: $-x^2$ shape – interval budgets follow this function with random perturbations. Little green power in the beginning, then supply with green energy is rising and falls at some point again (solar power from morning to evening, for example)

S2: x^2 shape – same situation as in S1, but starting from midday, again with random perturbations

Total of $2 \times 34 \times 4 \times 4 = 1088$ simulations per algorithm

Setup

- Different cluster sizes: `small` (72 nodes) and `large` (144 nodes)
- 34 real-world and synthetic workflows with $200 - 30\,000$ tasks
- Mapping generated with HEFT [Topcuoglu et al., IEEE TPDS, 2002], and baseline **ASAP** using EST, finishing in time D
- Different deadlines: D (tight deadline), $1.5D$, $2D$ and $3D$

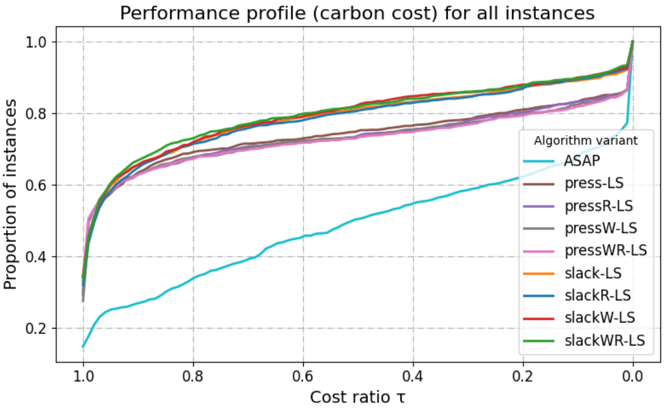
Different shapes for power profiles:

S3: $\sin(x)$ shape – 24 hours of this scenario. Little green power in the beginning and then sinus shape as given on $[0, 2\pi]$. We also add random perturbations

S4: Constant green power budget with perturbations – situations where one has storage for renewable energy or nuclear power

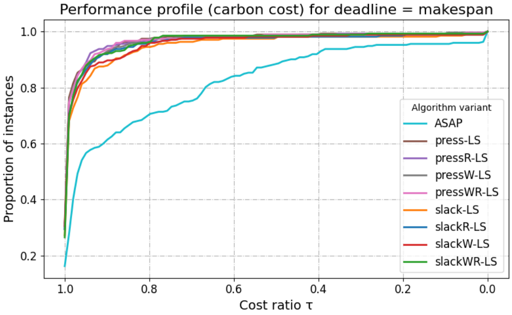
Total of $2 \times 34 \times 4 \times 4 = 1088$ simulations per algorithm

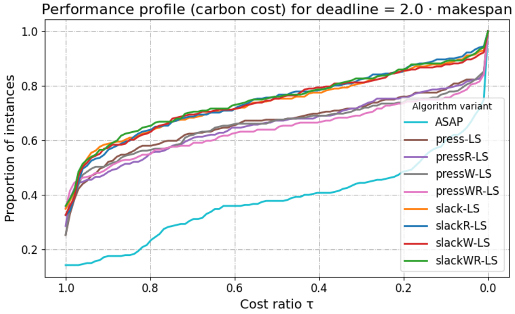
Evaluation – Including local search



- Cost ratio is best cost found divided by the algorithm variant's cost own cost – Higher curve is better
- Overall, slack variants seem to have the better curve
- Different situation depending on the deadline offset

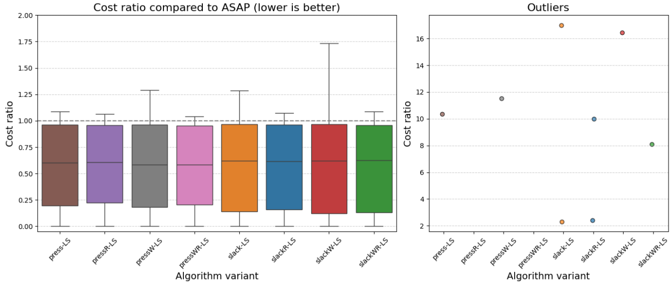
Evaluation – Including local search





- For tight deadlines, pressure variants perform better

Evaluation – Including local search



- Cost ratios obtained by dividing the heuristics carbon cost by the carbon cost of the baseline ASAP – Lower is better
- Scenarios with high green power at the beginning are good for the baseline
- Overall, median ratio around 0.6

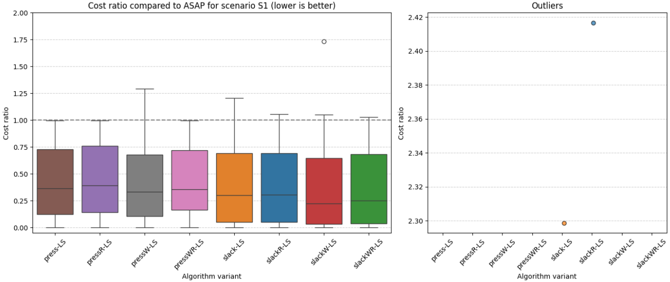
Evaluation – Influence of local search

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

- More than 400 simulations to evaluate the influence of local search
- Minimum, maximum and average cost ratios are shown
- By design, the cost ratio cannot get worse
- Cost ratio of 0 means, that by local search we found a 0-cost schedule while the initial solution has positive carbon cost

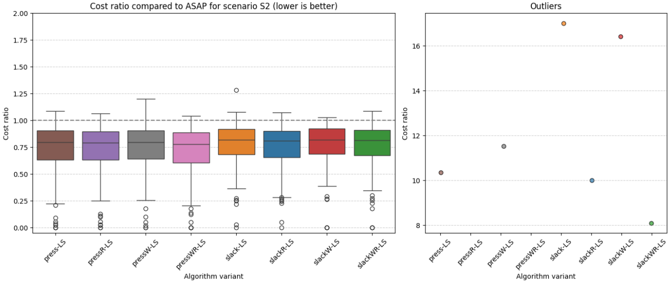
Solution improved by a factor $\approx 4.35\times$ better

Evaluation – Impact of parameters



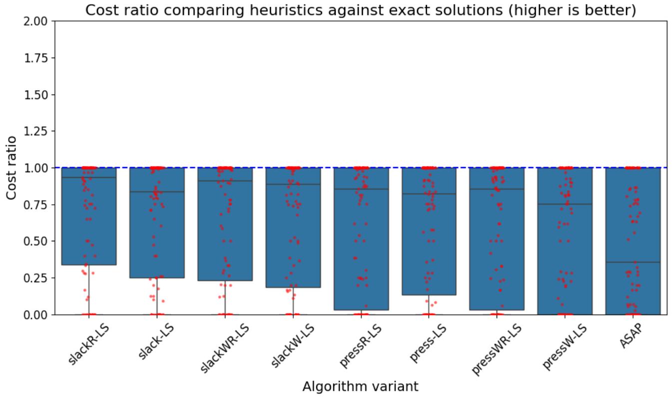
- The scenario influences the performance of the algorithms
- S1: Less green power budget in the beginning and towards end of the deadline
- Significant improvement with only few outliers

Evaluation – Impact of parameters



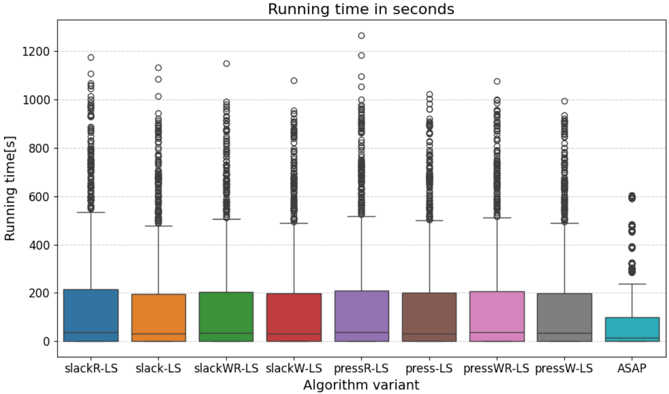
- The scenario influences the performance of the algorithms
- S2: A lot of green power in the beginning and towards end of the deadline
- Less improvement, but still significant
- More outliers: ASAP may take good decisions

Evaluation – Comparison with optimal solutions



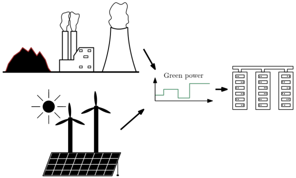
- Optimal solutions using an ILP formulation of the problem, Gurobi solver
- Only on small instances with up to 200 tasks
- A lot of instances are solved closed to optimality

Evaluation – Running time



- Aggregated running time values
- Compute a schedule within seconds; larger workflows (up to $30\,000$ tasks) can take several minutes
- Reasonable slowdown compared to baseline

Conclusion



- Minimizing carbon emissions when executing workflows on a parallel platform with a time-varying mix energy supply
- Focus on improving a given mapping by task shifting
- Theoretical analysis: DP algorithm for one processor, strong NP-completeness in simple case
- Several heuristics achieve significant savings in carbon emissions compared to ASAP baseline, close to optimal for small instances
- Major advances in understanding the problem of carbon-aware workflow scheduling

Future work: Carbon-aware extension of HEFT
First map and then optimize with fixed mapping