ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Intelligent Systems

HARDWARE DIMENSIONING FOR ENVIRONMENTAL SUSTAINABILITY: BENCHMARK OF AI ALGORITHMS AND ENVIRONMENTAL IMPACT

CANDIDATE
Enrico Morselli

SUPERVISOR
Prof. Andrea Borghesi

CO-SUPERVISOR
Prof. Allegra De Filippo

Academic year 2024-2025

Session 5th

dedicated(X) :- friend(X).

Contents

1 Introduction **1**

1.1 Background and Rationale 1

1.2 Introduction 1

2 Related Works **4**

2.1 Sustainability in AI 4

2.2 Tools for Tracking Carbon Emissions 6

3 Methodology **9**

3.1 Empirical Model Learning in HADA 9

3.1.1 Mathematical Formulation 10

3.1.2 Workflow of HADA 11

3.1.3 Extending HADA for Energy and Carbon Footprint Optimization 12

3.2 Carbon Footprint of Computation 12

3.2.1 Measuring Carbon Footprint with CodeCarbon 13

3.3 Extending HADA with CodeCarbon 15

3.3.1 Energy Management Algorithms 15

3.3.2 Incorporating Energy and Carbon Footprint Metrics . . 16

3.3.3 Min-Cut/Max-Flow Algorithms 17

4 Experimental Analysis **21**

4.1 Benchmarking on Different Hardware Platforms 21

4.1.1 Experimental Setup 21
4.1.2 Benchmarking ANTICIPATE and CONTINGENCY . 22
4.1.3 Benchmarking Min-Cut/Max-Flow Algorithms 23
4.1.4 Execution Constraints and Limitations 23
4.1.5 ANTICIPATE and CONTINGENCY 24
4.1.6 Min-Cut/Max-Flow Algorithms 32

5 HADA-as-a-Service **40**
5.1 HADA Web Application 40
5.1.1 Integration of Benchmark Data into HADA 44

6 Conclusions **47**

Bibliography **48**

Acknowledgements **53**

List of Figures

3.1 Carbon intensity of electricity generation, 2023. Source: [8] . 14

4.1 CO_2 emissions for ANTICIPATE and CONTINGENCY across different platforms. 25
4.2 Energy consumption for ANTICIPATE and CONTINGENCY across different platforms. 26
4.3 Correlation matrix for ANTICIPATE. 27
4.4 Correlation matrix for CONTINGENCY. 27
4.5 Time Comparison between ANTICIPATE and CONTINGENCY on leonardo and mbp19 30
4.6 Performance Comparison 32
4.7 CO2 emissions for Min-Cut/Max-Flow algorithms 33
4.8 Correlation matrix for BK 34
4.9 Correlation matrix for EIBFS 35
4.10 Correlation matrix for BK 35

5.1 The UI for the HADA web service application 41
5.2 Example of an optimisation request sent to HADA via the GUI 42
5.3 The solution provided for the above example in Figure 5.2 . . 43
5.4 Solution provided for the same constraint as in 5.2, but this time selecting ”Turkmenistan” as ”Country” 44
5.5 Organization of benchmark datasets for integration into HADA. 45

List of Tables

4.1 Experimental results for the ANTICIPATE algorithm with constraints on memory usage and solution cost. 29
4.2 Comparison of runtime statistics for the ANTICIPATE algorithm on different platforms. 29
4.3 Experimental results for the ANTICIPATE algorithm, minimizing runtime with constraints on solution cost and CO2 emissions 30
4.4 Runtime statistics for 96 scenarios using the ANTICIPATE algorithm. 31
4.5 Experimental results for the CONTINGENCY algorithm with constraints on runtime and solution cost. 31
4.6 CO2e(kg) statistics for different algorithms and platforms. . . 33
4.7 Experimental results for the BK algorithm 36
4.8 Experimental results for the EIBFS algorithm 37
4.9 Experimental results for the EIBFS algorithm 37
4.10 Experimental results for the BK algorithm 38
4.11 Experimental results for the EIBFS algorithm 38
4.12 Experimental results for the EIBFS algorithm 39

Chapter 1

Introduction

1.1 Background and Rationale

1.2 Introduction

In recent years, we have witnessed a dramatic improvement in the performance of Artificial Intelligence (AI) technologies. While AI still falls short of human ability in some complex cognitive tasks, as of 2023, it has surpassed human capabilities in various domains, including image classification, basic reading comprehension, visual reasoning, and natural language inference [20]. Moreover, Generative AI has achieved astonishing results in areas such as image and video generation, further demonstrating the rapid advancements in the field [20].

These remarkable gains in AI performance have been primarily driven by the large-scale expansion of model sizes and computational resources (often referred to as ”compute”) dedicated to training state-of-the-art AI models. Research indicates that, for frontier AI models—those that ranked among the top 10 in terms of training compute at the time of their release—the amount of compute used for training has been increasing at an exponential rate, growing by a factor of 4-5x per year since 2010 [24].

However, this surge in compute requirements has led to a corresponding increase in energy consumption, which, in turn, has amplified the environmental impact of AI due to CO_2 emissions. For instance, the training of Meta AI's LLaMA models was estimated to take approximately five months on a cluster of 2048 A100 80GB GPUs, consuming a total of $2,638$ MWh of energy and producing approximately $1,015$ tCO_2eq in emissions [27]. Given the growing adoption of AI technologies, the rapid increase in model size and complexity, and the escalating energy demands of AI applications, the carbon footprint of AI has become an increasingly pressing concern, particularly in the context of the ongoing climate emergency.

In this work, we explore a novel approach to addressing the sustainability challenges of AI by extending the HADA (HArdware Dimensioning for AI Algorithms) framework. HADA is a framework that leverages Machine Learning (ML) to learn the relationship between an algorithm's configuration and its performance metrics—such as total runtime, solution cost, and memory usage—and then employs optimization techniques to identify the optimal hardware architecture and configuration that meet predefined performance and budget constraints. This process, known as Hardware Dimensioning, has been studied as a means to efficiently allocate computational resources while maintaining performance requirements [6].

Our contribution extends HADA by incorporating energy consumption and carbon emissions as additional performance metrics. By doing so, we aim to identify the best combination of algorithm and hardware configuration that minimizes the environmental impact of computation. To validate our approach, we will conduct experiments on small-scale algorithms that can be executed in a timely manner on local machines and high-performance computing (HPC) clusters. Ultimately, we seek to demonstrate how AI workloads can be optimized not only for efficiency and cost but also for sustainability, thus contributing to the broader goal of reducing AI's carbon footprint.

The rest of the work is structured as follows:

- **Chapter 2** Reviews the environmental impact of AI, with case studies on the carbon footprint of training large models like GPT-3. It then discusses the concept of ”Green AI” vs. ”Red AI”—balancing accuracy with sustainability, and covers existing tools for tracking AI energy consumption and carbon emissions, such as ML CO2 Impact and CodeCarbon.

- **Chapter 3** Introduces the Empirical Model Learning (EML) approach, on which HADA is based on. Describes HADA's three-phase workflow: data collection, surrogate model training, and optimization. It then defines the methodology for tracking carbon emissions and integrates it into HADA's hardware dimensioning process.

- **Chapter 4** Presents benchmarking results for different AI algorithms, and analyzes energy consumption and CO2 emissions across different hardware platforms.

- **Chapter 5** Introduces the HADA Web Application, which provides a user-friendly interface for optimizing AI workloads. Describes how users can select algorithms, define optimization constraints, and choose geographic locations to assess energy efficiency.

- **Chapter 6** Summarizes key findings, emphasizing the importance of sustainability in AI computation and concludes the work.

Chapter 2

Related Works

2.1 Sustainability in AI

The environmental impact of training large AI models has been underscored by recent empirical assessments. Strubell et al. (2019) quantified the CO_2 emissions of several Natural Language Processing (NLP) models and discovered that training a large transformer with extensive hyperparameter tuning, including neural architecture search, emitted approximately 626,000 pounds (around 284 metric tons) of CO_2—comparable to the lifetime emissions of five cars [25]. Another study found out that GPT-3 released 552 metric tons of CO_2 into the atmosphere during training, the equivalent of 123 gasoline-powered passenger vehicles driven for one year [21]. These findings highlight that improvements in AI accuracy often come at a significant energy and carbon cost.

In response to these concerns, researchers have begun systematically reporting energy usage and the resulting CO_2 emissions associated with model training to raise awareness [7, 21]. Transparent reporting is essential for understanding and ultimately mitigating AI's climate impacts. For instance, Henderson et al. (2020) introduced a framework for tracking real-time energy consumption and carbon emissions during Machine Learning (ML) experiments, encouraging researchers to include these metrics in their publications [13].

Broader studies have examined AI's total energy and environmental footprint across the industry. Gupta et al. (2021) analyzed the end-to-end footprint of computing and found that while operational emissions (from running hardware) have been partly curbed by efficiency improvements, the overall carbon footprint of computing continues to grow due to increased scale. Notably, they showed that for modern data centers and mobile devices, manufacturing and infrastructure (embodied carbon) now account for the majority of emissions. As data centers adopt cleaner power, the emissions embedded in hardware supply chains—such as chip fabrication and server manufacturing—become a dominant concern. [12]

Similarly, Wu et al. (2022) conducted a comprehensive study of AI at a large technology company (Meta/Facebook), examining the entire AI model lifecycle—from data processing and training to inference and hardware lifecycle. They reported super-linear growth in AI workloads and infrastructure; for instance, daily inference operations doubled over a recent three-year span, necessitating a 2.5x expansion in server capacity. Crucially, Wu et al. also highlighted that embodied carbon is an increasing fraction of AI's total footprint, echoing that improvements in hardware efficiency alone cannot eliminate AI's impact. Their analysis argues for looking beyond training alone—considering data center construction, supply chains, and the frequency of model retraining—to truly grasp AI's environmental impact. [28]

A recurring theme in these studies is the diminishing return on energy investment for AI model improvements. As models become larger and more complex, incremental accuracy gains often require disproportionately more compute power, and thus energy. Schwartz et al. (2020) termed this phenomenon "Red AI," where researchers prioritize accuracy regardless of computational cost, and noted this trend is unsustainable both environmentally and economically [23]. Thompson et al. (2021) similarly observed that progress in benchmarks was accompanied by exponentially increasing computing costs, warning of diminishing returns and calling the situation unsustainable. [26]

Another challenge is equitable access: massive energy requirements make cutting-edge AI research expensive, potentially concentrating it in wealthy institutions and regions with robust infrastructure. This raises concerns that AI's growing energy demands not only harm the planet but also exacerbate inequalities in who can afford to conduct top-tier research. These concerns have prompted calls for a paradigm shift toward "Green AI," where efficiency and sustainability are treated as primary goals in model development.

In summary, the environmental sustainability of AI has become a critical area of concern. As the field advances, it is imperative to balance the pursuit of performance improvements with the need to minimize energy consumption and carbon emissions. This balance is essential not only for mitigating climate change but also for ensuring equitable access to AI research and development.

2.2 Tools for Tracking Carbon Emissions

The increasing awareness of AI's carbon footprint has motivated the development of dedicated tools and methodologies for monitoring the environmental impact of AI workloads. A number of open-source tools and frameworks have been introduced to help practitioners measure the energy consumption and CO_2 emissions of their code, enabling more informed decisions about sustainability in AI research and deployment.

One of the early tools for estimating emissions from model training was **ML CO2 Impact** (Machine Learning Emissions Calculator), introduced by Lacoste et al. (2019) [17]. This web-based calculator allows users to input key information about their training runs—such as hardware type (CPU/GPU), runtime, cloud provider, and location—and computes the corresponding energy consumption and carbon emissions. The tool relies on known power draw profiles of hardware components and regional carbon intensity factors to generate estimates. According to Bannour et al. (2021) [1], the latest iteration of this tool has evolved under the umbrella of the **CodeCarbon** initiative,

integrating its functionality into the open-source CodeCarbon package.

CodeCarbon [3], which is the tool we employ in this work to enhance HADA, is an open-source Python package designed to track the carbon footprint of computing projects. It integrates seamlessly into ML workflows, logging resource usage (CPU, GPU, and other components) and estimating the CO_2 emissions produced by the workload. A key feature of CodeCarbon is its ability to account for the geographic location of the computation—leveraging region-specific electricity carbon intensity data to provide location-dependent emission estimates. **Carbon Intensity** refers to the amount of CO_2 emitted per unit of electricity generated, typically measured in grams of CO_2 per kilowatt-hour (gCO_2/kWh). This metric varies significantly across regions depending on the energy mix used for electricity generation. For example, a region relying primarily on coal-fired power plants will have a much higher carbon intensity compared to one powered predominantly by renewable sources such as hydropower or wind energy. By incorporating this factor, CodeCarbon enables users to assess and compare the emissions impact of running their workloads in different locations. This means that shifting computations to regions with lower-carbon grids can lead to substantial reductions in total CO_2 emissions. This feature allows researchers and engineers to assess the impact of their computational choices; for example, running the same training job on a hydro-powered grid (e.g., in Montreal) results in significantly lower emissions than executing it on a coal-reliant grid. By making these comparisons explicit, CodeCarbon aims to inform and incentivize practitioners to optimize or relocate their workloads in a manner that reduces emissions.

Considerable efforts have also been made to quantify the carbon footprint of **Large Language Models (LLMs)**, as these models have experienced the most dramatic growth in both complexity and size. While much attention has been given to the energy demands of training these models, recent studies have also highlighted the substantial energy costs incurred during inference—the phase where models generate responses based on input data. Husom et

al. (2024) introduced MELODI (Monitoring Energy Levels and Optimization for Data-driven Inference), a comprehensive framework designed to monitor and analyze energy consumption during LLM inference processes [15]. MELODI enables detailed observations of power consumption dynamics and facilitates the creation of a dataset reflective of energy efficiency across varied deployment scenarios. Their findings indicate substantial disparities in energy efficiency, suggesting ample scope for optimization and the adoption of sustainable measures in LLM deployment. Similarly, Samsi et al. (2023) conducted experiments to study the computational and energy utilization of LLM inference, focusing on different sizes of LLaMA—a state-of-the-art LLM developed by Meta AI—on two generations of popular GPUs (NVIDIA V100 and A100) [22]. They benchmarked inference performance and energy costs across diverse tasks and presented results of multi-node, multi-GPU inference using model sharding across up to 32 GPUs. Their work provides valuable insights into compute performance and energy utilization characteristics of LLM inference, highlighting the need for cost-saving measures, efficient hardware usage, and optimal deployment strategies. Faiz et al. (2024) [9], introduces a model to estimate the total emissions of an LLM, accounting for both its *Operational* Carbon Footprint (energy consumption during training and inference) and its *Embodied* Carbon Footprint (emissions from hardware production and infrastructure). This study also highlights the importance of **Data Center Energy Efficiency**, measured by the *Power Usage Effectiveness (PUE)*, and the **Carbon Intensity of Electricity**—factors that reinforce the findings of CodeCarbon. Specifically, the study emphasizes that performing AI computations in regions where energy sources are cleaner (e.g., renewable-powered grids) can significantly reduce CO_2 emissions.

The development and adoption of these tracking tools play a crucial role in promoting sustainable AI research and deployment. By quantifying emissions and identifying optimization strategies, these tools provide the foundation for a more environmentally responsible approach to AI development.

Chapter 3

Methodology

3.1 Empirical Model Learning in HADA

As introduced earlier, the core focus of this work is **Hardware Dimensioning**, which refers to the challenge of identifying the most suitable hardware configuration for executing a given algorithm while meeting predefined performance and budget constraints. Specifically, given a target algorithm, our goal is to determine the optimal combination of algorithmic hyperparameters and hardware architecture that ensures the algorithm operates efficiently under specific performance requirements. However, this is a non-trivial optimization problem, primarily due to the difficulty of predicting an AI algorithm's performance across different hardware architectures and evaluating the impact of varying algorithmic and hardware configurations.

The key idea behind HADA is to integrate expert domain knowledge, such as execution time constraints, required solution quality, and budget limitations, with data-driven models that can infer performance relationships. At the foundation of HADA lies the **Empirical Model Learning (EML)** framework [19], which employs Machine Learning (ML) techniques to construct a model of an optimization problem. Instead of explicitly formulating complex relationships between algorithm performance and hardware resources,

EML leverages data-driven approximations, learning these dependencies empirically. These learned models are then embedded directly within the optimization process.

This approach is particularly advantageous in scenarios where deriving an exact mathematical formulation for the problem is infeasible due to its complexity. Broadly, EML facilitates the resolution of declarative optimization models that involve an intricate component, denoted as h, which captures the relationship between *decision variables* (also called *decidables*) x and *observables* y—i.e., measurable performance metrics of the system. This relationship is represented by the function $h(x) = y$. Since h is often highly complex and not directly optimizable, an alternative approach is to learn an approximate model, denoted as h_θ, where θ represents the set of learned parameters.

3.1.1 Mathematical Formulation

The general mathematical formulation of the EML framework can be expressed as follows:

$$\min \quad f(x, y) \tag{3.1}$$

$$\text{s.t.} \quad h_\theta(x) = y \tag{3.2}$$

$$g_j(x, y) \quad \forall j \in J \tag{3.3}$$

$$x_i \in D_i \quad \forall x_i \in x \tag{3.4}$$

where:

- x represents the vector of decision variables, each x_i belonging to a domain D_i;
- y denotes the vector of observed variables;
- The objective function $f(x, y)$, which depends on both decision and observed variables, is to be minimized;

- Constraints $g_j(x, y)$ enforce feasibility conditions on the variables, which can include classical mathematical programming inequalities and combinatorial constraints from Constraint Programming;
- The function $h_\theta(x)$ serves as an approximation of the complex relationship between decision and observed variables, and it is instantiated as a trained ML model.

The surrogate model $h_\theta(x)$ is learned through a standard supervised learning procedure. Given a training set $\mathcal{S} = \{(x_i, h(x_i))\}_{i=1}^m$, the goal is to determine the parameter vector θ that minimizes a loss function L:

$$\arg\min_\theta \frac{1}{m} \sum_{i=1}^{m} L(h_\theta(x_i), y_i^*) \tag{3.5}$$

where y_i^* are the ground-truth values (i.e., target outputs for regression tasks) and L is a predefined loss function, such as the L_1 or L_2 loss, depending on the nature of y.

3.1.2 Workflow of HADA

The HADA framework follows a structured three-phase approach:

1. **Data Collection (Benchmarking)**: In this initial phase, multiple runs of the target algorithm are conducted across different configurations, varying both hyperparameters and hardware settings. The purpose is to gather empirical performance data that will serve as the basis for training surrogate models.
2. **Surrogate Model Training**: Once the dataset $\mathcal{S}$ is constructed, ML models are trained on the collected data to learn the performance patterns. These models approximate the function $h_\theta(x)$ and are subsequently encoded into a structured optimization problem following the EML paradigm.

3. **Optimization**: The final phase involves solving the optimization problem, where user-defined constraints and an objective function are applied on top of the learned surrogate models and expert-defined constraints. The solver searches for the best combination of algorithmic and hardware configurations that meet the specified constraints while minimizing the objective function (e.g., energy consumption or execution time).

3.1.3 Extending HADA for Energy and Carbon Footprint Optimization

The core functionalities of *Surrogate Model Training* and *Optimization* are already implemented in the existing HADA prototype, which will be detailed in Chapter 5. Our primary contribution in this work is to extend HADA by incorporating additional metrics—**Resource Utilization, Energy Consumption, and Carbon Emissions**—into the empirical modeling process.

To achieve this, we systematically execute a variety of algorithms under diverse configurations (both in terms of hyperparameters and hardware platforms) while recording key performance metrics. These metrics serve as the target variables for the surrogate models, allowing us to model the energy footprint and environmental impact of computational workloads. Before implementing this extension, however, it is necessary to establish a framework for accurately measuring the Carbon Footprint of an AI algorithm. This is the focus of the next section.

3.2 Carbon Footprint of Computation

The carbon footprint of an algorithm is determined by two main factors: (1) the energy required to execute it and (2) the emissions produced in generating that energy. The first factor depends on computing resource usage—such as

the number of processing cores, execution time, and data center efficiency—while the second, known as **Carbon Intensity**, is influenced by the energy production methods and geographic location.

Carbon Intensity quantifies the greenhouse gas (GHG) emissions associated with electricity production, expressed in terms of carbon dioxide equivalent (CO_2e). This metric accounts for the global warming potential of various GHGs emitted over a given timeframe [18]. As described in [3], the total carbon footprint of a computational task can be estimated using the following equation:

$$C = E \times CI \tag{3.6}$$

where:

- E represents the total energy consumed by the computational infrastructure, measured in kilowatt-hours (kWh).
- CI represents the carbon intensity of the electricity consumed, measured in grams of CO_2e per kilowatt-hour (gCO_2e/kWh).

3.2.1 Measuring Carbon Footprint with CodeCarbon

CodeCarbon is an open-source tool designed to estimate the carbon footprint of computational workloads. It achieves this by directly measuring energy consumption from key hardware components—CPU, GPU, and RAM—at regular intervals (default: every 15 seconds). Additionally, it tracks execution time to compute the total electricity usage of the system.

A distinguishing feature of CodeCarbon is its ability to incorporate region-specific **Carbon Intensity** data to provide location-dependent emission estimates. Carbon intensity is computed as a weighted average of the emissions from various energy sources used to generate electricity. These sources can be categorized into:

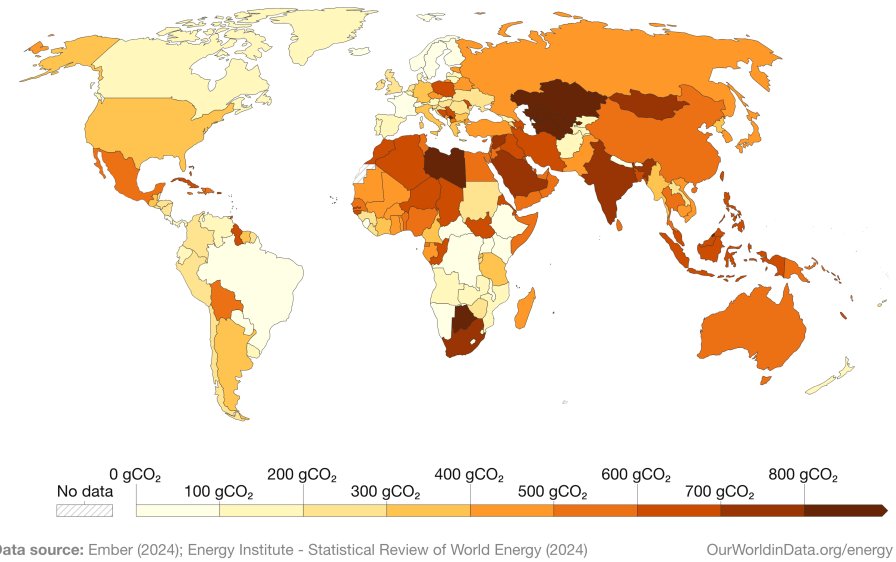


Figure 3.1: Carbon intensity of electricity generation, 2023. Source: [8]

- **Fossil Fuels**: Coal, petroleum, and natural gas—each associated with specific carbon intensities, meaning a known quantity of carbon dioxide is emitted per kilowatt-hour of electricity generated.
- **Renewable and Low-Carbon Energy**: Solar, wind, hydroelectricity, biomass, geothermal, and nuclear power—sources with significantly lower or near-zero carbon emissions.

Depending on the platform and data availability, CodeCarbon retrieves carbon intensity information from various sources. For private infrastructures, it relies on datasets such as those provided by Our World In Data [8]. Figure 3.1 illustrates the carbon intensity of electricity generation by country as of 2023.

By integrating real-time hardware usage monitoring with location-based carbon intensity data, CodeCarbon enables researchers and practitioners to quantify the environmental impact of their computations. This information can help guide decision-making, such as optimizing code efficiency, choosing lower-carbon data centers, or scheduling workloads during periods of higher renewable energy availability.

3.3 Extending HADA with CodeCarbon

3.3.1 Energy Management Algorithms

The first step in extending HADA to incorporate Carbon Intensity is to construct a dataset that includes carbon footprint data from algorithm execution. This requires integrating **CodeCarbon** into the computational workflow used for running algorithms.

For this study, we analyzed two algorithms: **ANTICIPATE** and **CONTINGENCY**, originally introduced in [4, 5] and previously used in the HADA framework paper [6]. These algorithms belong to the domain of **Energy Management Systems** and are designed to compute the amount of energy that must be generated to meet a required load while minimizing the total energy cost over a daily time horizon. Both methods consider the inherent uncertainty in energy demand and production.

- **ANTICIPATE** is an *online*, scenario-based anticipatory algorithm that adapts dynamically to uncertainty.
- **CONTINGENCY** is a hybrid *offline/online* approach that first constructs (offline) a pool of solutions, which is then used to guide an efficient online decision-making process.

Both methods are characterized by a tunable configuration parameter that can be adjusted based on performance constraints, particularly execution time and solution quality:

- For **ANTICIPATE**, the configuration parameter is the number of considered *scenarios* (`nScenarios`).
- For **CONTINGENCY**, it is the number of *traces* (`nTraces`).

The goal of HADA is to learn the relationship between this configuration parameter and various performance metrics that characterize algorithm efficiency and resource usage. The original dataset used in HADA includes the

following attributes:

- `nParam`: The configuration parameter of the algorithm (`nScenarios` for ANTICIPATE, `nTraces` for CONTINGENCY).
- `time(s)`: Execution time required to find a solution.
- `sol(keuro)`: Cost of the obtained solution (expressed in thousand euros), serving as a measure of solution quality.
- `memAvg(MB)`: Average memory usage of the algorithm (in MB).

3.3.2 Incorporating Energy and Carbon Footprint Metrics

To extend this approach, we integrated additional metrics related to **energy consumption** and **carbon footprint**, leveraging the capabilities of CodeCarbon. The following attributes were added to the dataset:

- `emissions`: Total CO_2e emissions generated during execution (kg).
- `emission_rate`: Rate of CO_2e emissions per second (kg/s).
- `cpu_energy`: Energy consumed by the CPU (kWh).
- `ram_energy`: Energy consumed by RAM (kWh).
- `tot_energy`: Total energy consumption (kWh), which is the sum of `cpu_energy` and `ram_energy`. If applicable, it also includes GPU energy consumption.
- `country` and `region`: Geographic location where the computation took place, used to determine the carbon intensity of electricity.
- `cpu_count`: Number of CPU cores used.

Additionally, we implemented tracking for the peak memory usage of each execution, which we included in the dataset under the attribute:

- `memPeak(MB)`: Maximum memory usage (in MB) recorded during execution.

By incorporating these additional metrics, HADA can now model not only algorithm efficiency and solution quality but also the environmental impact of computational workloads. This enhancement allows for a more comprehensive optimization approach, balancing performance requirements with sustainability considerations.

3.3.3 Min-Cut/Max-Flow Algorithms

Following our initial experiments with ANTICIPATE and CONTINGENCY, which were already included in HADA, we sought to extend our study to a broader class of algorithms. We selected a set of algorithms used to solve the **Minimum Cut/Maximum Flow (Min-Cut/Max-Flow)** problem in graphs, a fundamental problem in combinatorial optimization with widespread applications in various domains, particularly in **Computer Vision**.

The Min-Cut/Max-Flow problem involves computing the maximum amount of flow that can be sent from a designated source node to a ink node in a flow network, subject to capacity constraints on edges. The **Max-Flow** problem seeks to maximize the total flow, while the **Min-Cut** problem finds the smallest set of edges that, if removed, would disconnect the source from the sink. The Max-Flow problem and the Min-Cut problem are closely related, as stated by the **Max-Flow Min-Cut Theorem**, which asserts that the maximum flow value in a network is equal to the capacity of the minimum cut.

Min-Cut/Max-Flow algorithms are widely used in image segmentation, stereo vision, and other Computer Vision tasks where images are represented as graphs, and energy minimization techniques are applied to solve labeling problems. We referred to the work of Jensen et al. (2023) [16], which provides a comprehensive review of state-of-the-art Min-Cut/Max-Flow algorithms, evaluated on a large dataset of Computer Vision problems.

Selected Algorithms

Based on the findings of Jensen et al., we focused on three well-known Min-Cut/Max-Flow algorithms, each representing different families of flow-based optimization methods:

- **Boykov-Kolmogorov (BK)**
- **Excess Incremental Breadth First Search (EIBFS)**
- **Hochbaum's Pseudo-Flow (HPF)**

According to Jensen et al., Min-Cut/Max-Flow algorithms can be broadly classified into different families based on their approach to augmenting paths and flow computation. The **Boykov-Kolmogorov** (BK) algorithm [2] belongs to the **Augmenting Paths (AP)** family, which is one of the oldest methods for solving the Max-Flow problem. This family dates back to the classical **Ford-Fulkerson Algorithm** [10], which introduced the concept of augmenting paths to iteratively increase the flow. BK extends this approach by introducing heuristic techniques to improve efficiency, making it particularly effective for Computer Vision applications.

On the other hand, **Excess Incremental Breadth First Search (EIBFS)** [11] and **Hochbaum's Pseudo-Flow (HPF)** [14] belong to the **Pseudoflow** family, which differs from augmenting path methods in the way it manages flow excess. These algorithms prioritize maintaining a valid preflow at all times and use alternative strategies to push flow across the network, often improving efficiency in large-scale instances.

The main differences between these algorithm families lie in the order in which they traverse nodes while searching for augmenting paths and in their mechanisms for pushing flow along paths in the graph.

Algorithm Variants Considered

Unlike ANTICIPATE and CONTINGENCY, which have explicit tunable configuration parameters, these Min-Cut/Max-Flow algorithms can be implemented with different optimizations [16]. We treated the specific implementation variant of each algorithm as a configuration parameter and considered the following versions:

- **Boykov-Kolmogorov (BK) Variants:**
  - **BK**: The reference implementation.
  - **MBK**: An optimized version by Jensen et al., using indices instead of pointers to reduce memory footprint.
  - **MBK-R**: A second optimized version that reorders arcs to ensure that all outgoing edges from a node are stored contiguously in memory.
- **Excess Incremental Breadth First Search (EIBFS) Variants:**
  - **EIBFS**: A slightly modified version of the original EIBFS algorithm.
  - **EIBFS-I**: A version that replaces pointers with indices to improve memory locality.
  - **EIBFS-I-NR**: A version similar to EIBFS-I but without arc reordering.
- **Hochbaum's Pseudo-Flow (HPF) Variants:**
  - **HPF-H-F**: Highest-label variant using FIFO buckets.
  - **HPF-H-L**: Highest-label variant using LIFO buckets.
  - **HPF-L-F**: Lowest-label variant using FIFO buckets.
  - **HPF-L-L**: Lowest-label variant using LIFO buckets.

Energy and Carbon Footprint Monitoring

Jensen et al. provided scripts to evaluate the runtime performance of these algorithms, measuring both initialization time and execution time. To extend this analysis with energy consumption and carbon footprint tracking, we modified their experimental setup by integrating **CodeCarbon**.

Since the original implementations are written in C++, we wrapped the algorithm scripts with a Python interface to facilitate real-time monitoring. This wrapper was designed to:

- Track memory usage throughout execution.
- Use CodeCarbon to measure CPU and RAM energy consumption.
- Record CO_2e emissions based on the energy consumption and Carbon Intensity of the execution environment.

Unlike the experiments with ANTICIPATE and CONTINGENCY, where solution quality was a critical performance metric, here we focus solely on computational efficiency. The primary metrics of interest for this study are execution time, memory consumption, energy consumption, and environmental impact. By incorporating these Min-Cut/Max-Flow algorithms into HADA, we aim to explore energy-efficient graph-based optimization techniques and further refine our approach to sustainable AI computation.

Chapter 4

Experimental Analysis

4.1 Benchmarking on Different Hardware Platforms

To comprehensively evaluate the performance of the selected algorithms under varying computational environments, we conducted experiments on multiple hardware platforms. This approach allows us to better capture how changes in hardware configurations impact execution time, energy consumption, and carbon footprint.

4.1.1 Experimental Setup

The benchmark phase was carried out on the following computing environments:

- `mbp19`: A personal laptop running MacOS Ventura 13.6.7, equipped with:
  - 1.4 GHz Quad-Core Intel Core i5 processor;
  - 8 GB 2133 MHz LPDDR3 RAM.
- `PC`: A desktop computer running Ubuntu 24.4 LTS, featuring:

  - 3.4 GHz Intel Core i7-4770 Quad-Core processor;
  - 8 GB DDR3 RAM.

- `leonardo`: A high-performance computing (HPC) infrastructure, **Leonardo**, hosted by **CINECA**. Leonardo consists of 4992 computing nodes, distributed across two primary partitions, i.e. the Booster Partition and the Data Centric General Purpose partition. For our benchmark, we will rely on the booster partition, which has the following specifics:
  - 512 GB (8 × 64 GB DDR4 3200 MHz) RAM per node;
  - Single-socket, 32-core Intel Xeon Platinum 8358 CPU, 2.60 GHz (Ice Lake) (110,592 total cores);
  - 4× NVIDIA custom Ampere A100 GPUs with 64 GB HBM2e memory, NVLink 3.0 (200 GB/s).

Leonardo offers the possibility of launching jobs for execution on the booster partition using the slurm job scheduler, by specified the required runtime and resources. For the benchmark runs with anticipate and contingency, we requested 16 cores, while the min-cut/max-flow algorithm runs were executed on 32 cores.

4.1.2 Benchmarking ANTICIPATE and CONTINGENCY

For the first set of experiments, we executed **ANTICIPATE** and **CONTINGENCY** on a collection of 30 problem instances. These instances were sampled using a Gaussian statistical model, as proposed in [6].

Each algorithm was executed on every instance with a varying configuration parameter (i.e., the number of *traces* for CONTINGENCY and *scenarios* for ANTICIPATE), ranging from 1 to 100. These experiments were performed on both `mbp19` and `leonardo`. As a result, we generated a dataset containing:

$$30 \times 100 \times 2 \times 2 = 12,000 \text{ entries.} \tag{4.1}$$

This dataset captures execution time, memory consumption, energy usage, and carbon footprint across different algorithmic configurations and hardware platforms.

4.1.3 Benchmarking Min-Cut/Max-Flow Algorithms

To evaluate the performance of the selected **Min-Cut/Max-Flow** algorithms, we relied on the dataset provided by Jensen et al. (2023) [16]. This dataset contains a diverse set of Min-Cut/Max-Flow problem instances from the field of Computer Vision, allowing for a thorough assessment of algorithmic efficiency in real-world applications. Jensen et al. also provided reference implementations of the algorithms, which we used for benchmarking.

Since these implementations were written in **C++** (for BK and EIBFS) and **C** (for HPF), integrating energy consumption tracking posed a challenge. Currently, few tools are available for directly monitoring the energy footprint of compiled C/C++ code.

To address this issue, we developed a **Python wrapper script** that executes the C++/C implementations and integrates **CodeCarbon** for energy and emission tracking. The wrapper allows us to monitor:

- Execution time and memory usage;
- CPU and RAM energy consumption;
- CO_2e emissions based on real-time energy monitoring and Carbon Intensity data.

4.1.4 Execution Constraints and Limitations

While our goal was to run the selected algorithms on all instances of the Jensen et al. dataset, several factors constrained the scope of our experiments:

- **Compilation and Platform Limitations:** The BK and EIBFS implementations could not be compiled on `mbp19` due to missing dependencies and a lack of support for MacOS. To address this, we utilized an alternative system—`PC`—running a lightweight Ubuntu distribution, enabling execution on additional hardware.

- **Memory Constraints:** Some problem instances in the dataset were too large to fit within the available system memory. Although it was technically possible to use a swap partition to accommodate larger instances, this would have distorted memory usage measurements, as available monitoring tools record only RAM consumption and do not account for swapped memory.

- **Execution Time Considerations:** Given the extensive number of instances in the dataset, running all of them would have been prohibitively time-consuming. As a result, we selected a representative subset of problem instances that allowed us to evaluate performance across a range of complexity levels without exceeding computational constraints.

By executing experiments across multiple hardware platforms and addressing challenges related to execution and monitoring, we constructed a dataset that provides insights into algorithmic efficiency, energy consumption, and carbon emissions. The following sections will analyze the results obtained from these experiments and explore the implications of integrating sustainability metrics into algorithm selection and hardware dimensioning.

4.1.5 ANTICIPATE and CONTINGENCY

Following the completion of the benchmark phase, we analyze the resulting data to evaluate the carbon emissions and energy consumption of the ANTICIPATE and CONTINGENCY algorithms. This section presents a comparative analysis of their environmental impact across different hardware platforms.

Carbon Emissions Analysis

Figure 4.1 displays the carbon emissions for ANTICIPATE and CONTINGENCY, categorized by hardware platform. The data reveals that emissions tend to be higher for the ANTICIPATE algorithm, particularly when executed on the `leonardo` platform. Overall, the measured values remain relatively small, except for one notable outlier exceeding 0.020 kg of CO_2e. Most emission values fall within the range $[0, 0.005]$ kg.

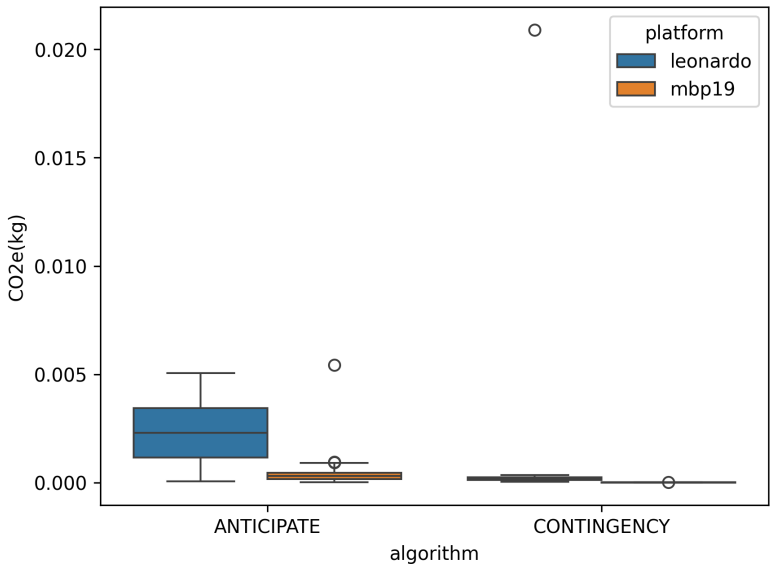


Figure 4.1: CO_2 emissions for ANTICIPATE and CONTINGENCY across different platforms.

Energy Consumption Analysis

Similarly, Figure 4.2 illustrates the energy consumption for ANTICIPATE and CONTINGENCY. The overall trend aligns with the emissions data, with ANTICIPATE exhibiting higher energy consumption than CONTINGENCY, particularly on `leonardo`, given that carbon emissions are directly proportional to energy usage in the absence of variation in Carbon Intensity.

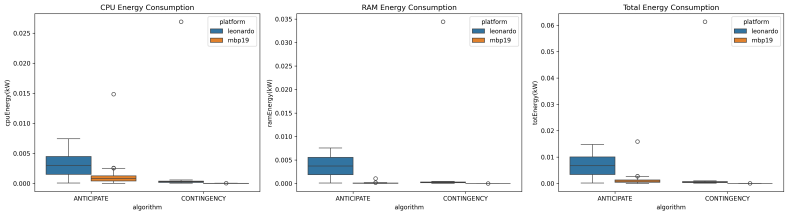


Figure 4.2: Energy consumption for ANTICIPATE and CONTINGENCY across different platforms.

Correlation Analysis

To further investigate the factors influencing emissions and energy consumption, Figures 4.3 and 4.4 present the correlation matrices for ANTICIPATE and CONTINGENCY, respectively.

For ANTICIPATE (Figure 4.3), the variable exhibiting the highest correlation with carbon emissions, aside from direct energy consumption, is the **average memory usage**, with a correlation coefficient of 0.93. This suggests that higher memory consumption is a key contributor to increased emissions. Additionally, CPU energy consumption and RAM energy consumption both exhibit strong correlations with emissions, further reinforcing the relationship between computational resource usage and environmental impact.

In contrast, for CONTINGENCY (Figure 4.4), there are no particularly strong linear correlations between emissions and other performance metrics. The highest correlation with emissions is observed for **execution time** (0.36), indicating that longer runtimes slightly contribute to higher emissions. However, the overall relationships appear weaker compared to ANTICIPATE, suggesting that CONTINGENCY's energy usage is less dependent on a single dominant factor.

From the correlation analysis, we make the following key observations:

- Carbon emissions are perfectly proportional to total energy consumption, as indicated by the correlation of 1.0 between `totEnergy` and `CO2e`. This reflects the fact that, in the absence of varying Carbon Intensity, emissions are solely determined by energy consumption.

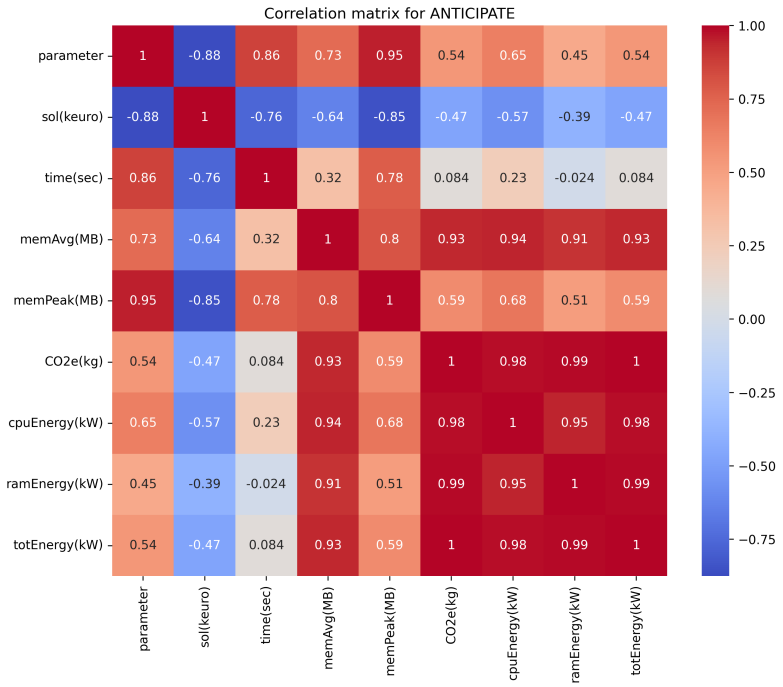


Figure 4.3: Correlation matrix for ANTICIPATE.

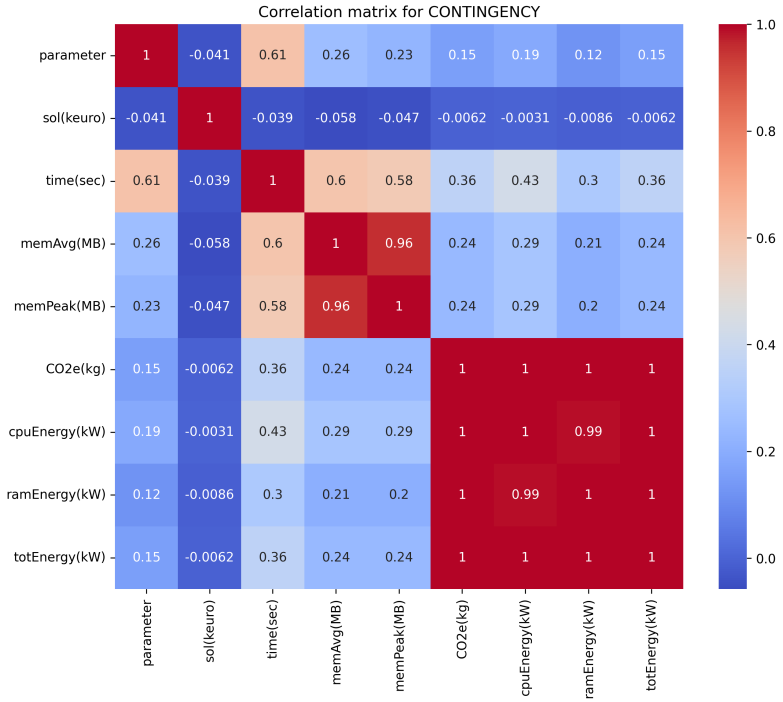


Figure 4.4: Correlation matrix for CONTINGENCY.

- Memory usage plays a significant role in determining emissions for ANTICIPATE, with a correlation of 0.93. This suggests that optimizing

memory allocation could be an effective s trategy for reducing emissions.

- For CONTINGENCY, execution time has the highest correlation with emissions (0.36), indicating a weaker but notable relationship.
- CPU and RAM energy consumption are highly correlated with total emissions (0.98 and 0.99, respectively). Since total energy consumption is simply the sum of CPU and RAM energy, this is expected.
- Retaining energy consumption values in the dataset is beneficial as these values can be scaled based on different Carbon Intensity factors. This enables an analysis of the impact of computing in different geographic locations.

Testing the HADA Framework with the Expanded Datasets

To further explore the optimization capabilities of HADA, we conducted additional experiments on ANTICIPATE and CONTINGENCY, incorporating constraints on runtime, memory usage, and solution cost. The goal was to determine whether HADA could effectively optimize carbon emissions while satisfying various computational constraints.

The results for ANTICIPATE are presented in Table 4.1. We tested multiple combinations of constraints on solution cost (expressed in thousands of euros, k€) and average memory consumption (expressed in MB), while minimizing CO_2 emissions (expressed in kg). In cases where HADA was unable to find a valid solution satisfying the constraints, we marked the corresponding results as ”none”.

The results indicate that in most cases, minimizing CO_2 emissions aligns with minimizing memory consumption. The smallest emissions were observed when HADA selected configurations with fewer scenarios (n^P). This is consistent with our earlier observations, where ANTICIPATE’s carbon footprint was highly correlated with memory usage (0.93 correlation). We can also

Bounds		Solution				
Memory (MB)	Cost (k€)	n^P	HW	CO_2 (kg)	Memory (MB)	Cost (k€)
no	no	1	mbp19	7.23×10^{-6}	-	-
80	no	none	none	none	none	none
no	100	none	none	none	none	none
100	300	none	none	none	none	none
150	300	64	mbp19	3.75×10^{-4}	141.53	299.46
170	270	97	mbp19	5.70×10^{-4}	169.85	267.39
170	280	87	mbp19	5.10×10^{-4}	160.12	276.86
300	500	1	mbp19	7.23×10^{-6}	81.47	368.73

Table 4.1: Experimental results for the ANTICIPATE algorithm with constraints on memory usage and solution cost.

observe how for tighter solution cost bounds, the numbber of scenarios tends to increase (reflecting the negative correlation of -0.88 as we can see in 4.3), and since increasing the number of scenarios means more memory usage, the carbon emissions also grows. When the solution cost is allowed to be higher, the number of scenarios is instead kept as low as possible, given that we are trying to minimize the CO_2 emissions. Thus, in the case of ANTICIPATE we have a trade-off between CO_2 emissions and Solution cost.

A notable outcome is that, when trying to minimize the emissions HADA always selected the mbp19 platform. This suggests that the additional computational power offered by leonardo comes with an higher energy cost, as shown by 4.1 and 4.2, so that, when the focus is on minimizing the CO_2 emissions mbp19 seems to be the ”greener” choice. The advanges of using leonardo may come instead from the reduced runtimes.

4.5 shows that, for ANTICIPATE, the execution time time on leonardo is smaller on average, which is confirmed by 4.2

Algorithm	**Platform**	**Mean**	**Std**	**Min**	**Max**
ANTICIPATE	leonardo	52.876531	31.205927	1.22	115.18
	mbp19	93.023027	57.730152	1.50	284.29

Table 4.2: Comparison of runtime statistics for the ANTICIPATE algorithm on different platforms.

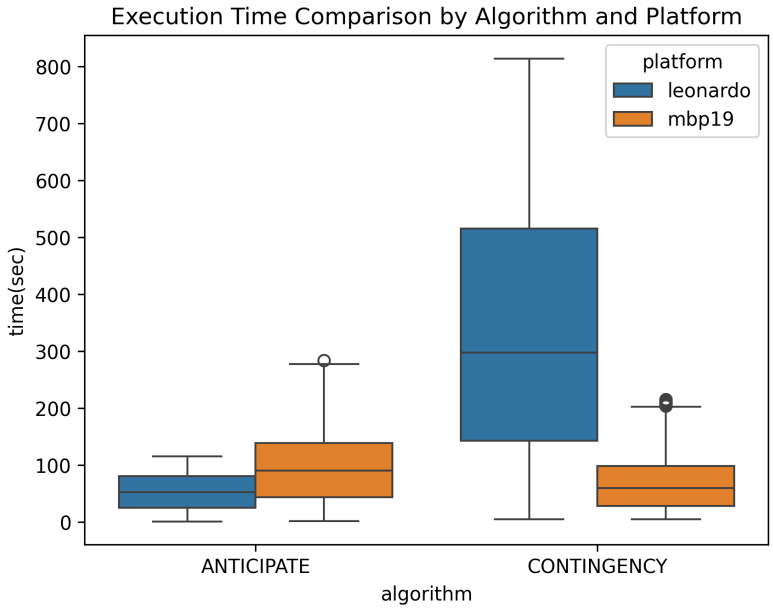


Figure 4.5: Time Comparison between ANTICIPATE and CONTINGENCY on leonardo and mbp19

We can try to test the potential benefits of leonardo by using the runtime as objective to minimize in HADA. 4.3 presents the results of a set of experiments with the objective of runtime minimization of ANTICIPATE, with constrinats on the solution cost and the CO_2 emissions.

Bounds		Solution				
CO2eq (kg)	Cost (k€)	n^P	HW	CO_2 (kg)	Time (s)	Cost (k€)
0.005	300	60	leonardo	2.94×10^{-3}	62.63	299.92
0.01	280	87	leonardo	4.24×10^{-3}	92.04	274.01
0.1	270	96	leonardo	4.74×10^{-3}	100.94	267.40
0.0001	400	1	leonardo	7.53×10^{-5}	1.24	368.73
0.00001	400	1	mbp19	7.23×10^{-6}	1.53	368.73

Table 4.3: Experimental results for the ANTICIPATE algorithm, minimizing runtime with constraints on solution cost and CO2 emissions

From these experiments we can see that when we try to minimize the runtime, and we have tight bounds on the solution cost, the preferrable choice is leonardo, with an high number of scenarios. So this confirms that leonardo can actually provide advantages when the emphasis is on runtime minimization and constraints on the solution cost, since to obtain cheaper solutions a

higher number of scenarios is required, and this cause an increase in the runtime. For example, we can compare the runtimes of ANTICIPATE with 96 scenarios on both platforms, and we can see that on mbp19 they are considerably higher (See 4.4):

Platform	Mean	Std	Min	Max
leonardo	101.663448	2.280052	96.47	106.40
mbp19	184.732667	23.545764	167.55	256.84

Table 4.4: Runtime statistics for 96 scenarios using the ANTICIPATE algorithm.

However, for the same configuration of the algorithm on mbp19, the emission values are an order of magnitude lower. Indeed, when constraints on the carbon emissions are very low (1×10^{-5} or less), HADA starts preferring mbp19 to leonardo, due to the greater energy efficiency for smaller values of the parameter, at the expense of an higher solution cost.

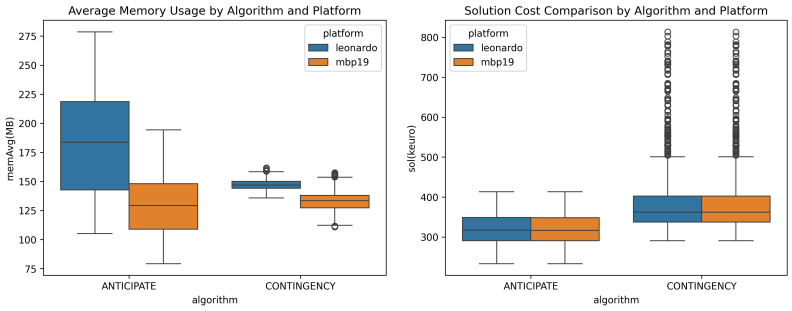
For CONTINGENCY, Table 4.5 presents the results of optimizing CO_2 emissions under constraints on execution time and solution cost. Given that execution time showed the highest correlation with emissions (0.47), we focused on determining whether limiting runtime would impact emissions optimization.

Bounds		Solution				
Time (s)	Cost (k€)	n^P	HW	CO_2 (kg)	Time (s)	Cost (k€)
no	no	2	mbp19	1.29×10^{-6}	-	-
no	350	4	mbp19	1.53×10^{-6}	-	$338.23k$
10	no	2	mbp19	1.29×10^{-6}	6.01	-
10	320	none	none	none	none	none
60	350	4	mbp19	1.29×10^{-6}	7.68	$338.23k$
120	320	90	mbp19	1.24×10^{-5}	119.47	$314.11k$

Table 4.5: Experimental results for the CONTINGENCY algorithm with constraints on runtime and solution cost.

The results for CONTINGENCY further reinforce the platform selection pattern seen in ANTICIPATE: HADA consistently favored mbp19 over Leonardo.

This was expected, as we can see In 4.5, that CONTINGENCY runs significantly slower on Leonardo than on mbp19. By looking at the dataset, we can see that Leonardo incurred a large overhead ($\approx$ 60s) even for a single trace, whereas mbp19 handled one trace in 5-6s. This trend continued as traces increased, as with 100 traces, mbp19 has runtimes around 140 seconds, while leonardo runtime values are around 800 seconds. Thus, contrary to what happened with ANTICIPATE, with CONTINGENCY using leonardo did not brought any significant advantage in performance. Even by looking at the memory consumption we can see that mbp19 outperforms leonardo(4.6a), with actually no visible gains in terms of solution cost, as we can see by 4.6b.



(a) Average memory usage (b) Cost comparison

Figure 4.6: Performance Comparison

4.1.6 Min-Cut/Max-Flow Algorithms

In this section, we analyze the performance and environmental impact of Min-Cut/Max-Flow algorithms and discuss the results obtained from tests performed using the HADA framework. These algorithms play a crucial role in various optimization problems, particularly in computer vision and network flow applications. Figure 4.7 presents an overview of the CO_2 emissions for the three Min-Cut/Max-Flow algorithms tested. It is evident that the vast majority of configurations emitted less than 0.001 kg of CO_2, similar to trends observed for the ANTICIPATE and CONTINGENCY algorithms.

Carbon Emissions Analysis

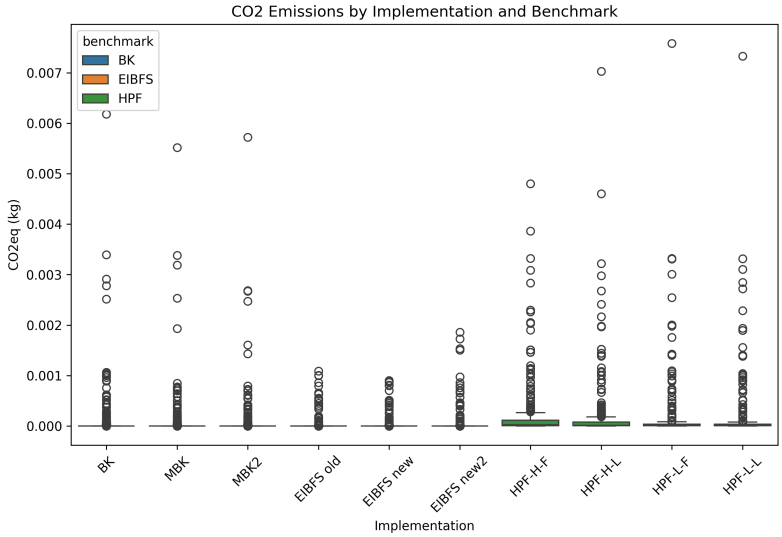


Figure 4.7: CO2 emissions for Min-Cut/Max-Flow algorithms

By examining emissions data in Table 4.6, we confirm that smaller architectures, such as `pc` and `mbp19`, tend to be more energy-efficient than the high-performance computing (HPC) system Leonardo. However, it is important to note that not all problem instances were executed on every platform due to constraints on memory availability and computational feasibility.

Algorithm	Platform	Mean	Min	Max
BK	leonardo	5.155456×10^{-5}	2.883260×10^{-6}	0.006184
	pc	8.010042×10^{-7}	8.684843×10^{-9}	0.000145
EIBFS	leonardo	2.364746×10^{-5}	2.897591×10^{-6}	0.001861
	pc	5.059142×10^{-7}	9.001983×10^{-9}	0.000027
HPF	leonardo	2.283886×10^{-4}	2.960032×10^{-5}	0.007584
	mbp19	1.232039×10^{-5}	4.693739×10^{-7}	0.001145

Table 4.6: CO2e(kg) statistics for different algorithms and platforms.

The discrepancy in emissions can largely be attributed to the scale of the problem instances executed on each platform. Due to the significantly larger memory available on Leonardo, the most demanding instances (with node counts reaching hundreds of millions) were exclusively executed on this HPC system.

Additionally, some problem instances could not be solved by HPF implementations due to their reliance on negative capacity arcs. While BK and EIBFS supported such instances, HPF's current implementation does not, resulting in the exclusion of these cases from the dataset.

Correlation Analysis

To further understand the factors influencing emissions, Figures 4.8, 4.9, and 4.10 present the correlation matrices for BK, EIBFS, and HPF, respectively.

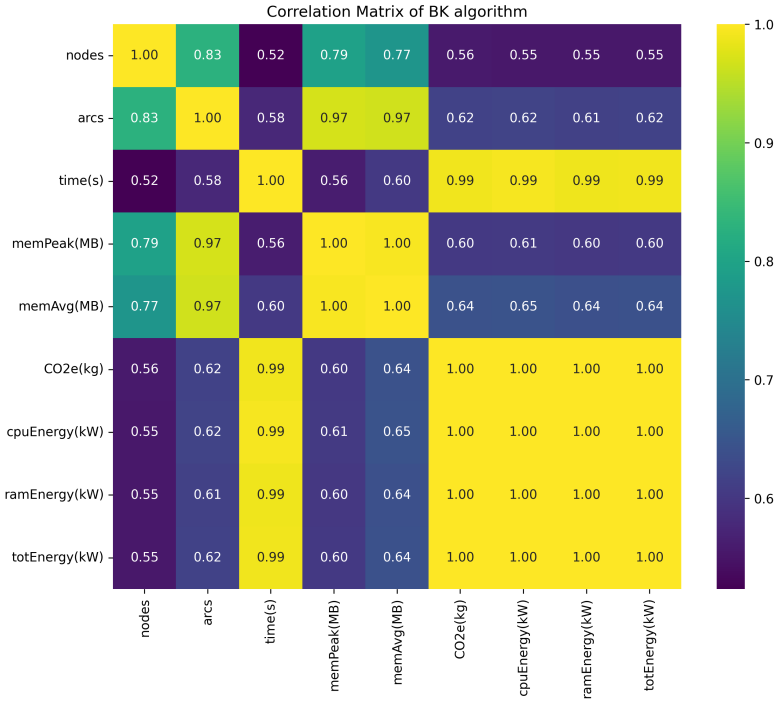


Figure 4.8: Correlation matrix for BK

For both BK and EIBFS, runtime exhibits the strongest correlation with CO_2 emissions. In 4.8, we observe that execution time has a correlation of approximately 0.99 with energy consumption and emissions, indicating that the longer an algorithm runs, the more energy it consumes, which directly translates into higher emissions. This trend is similarly observed in EIBFS (4.9), reinforcing that for these algorithms, minimizing runtime is the most effective way to reduce emissions. Given this insight, we will assess how BK

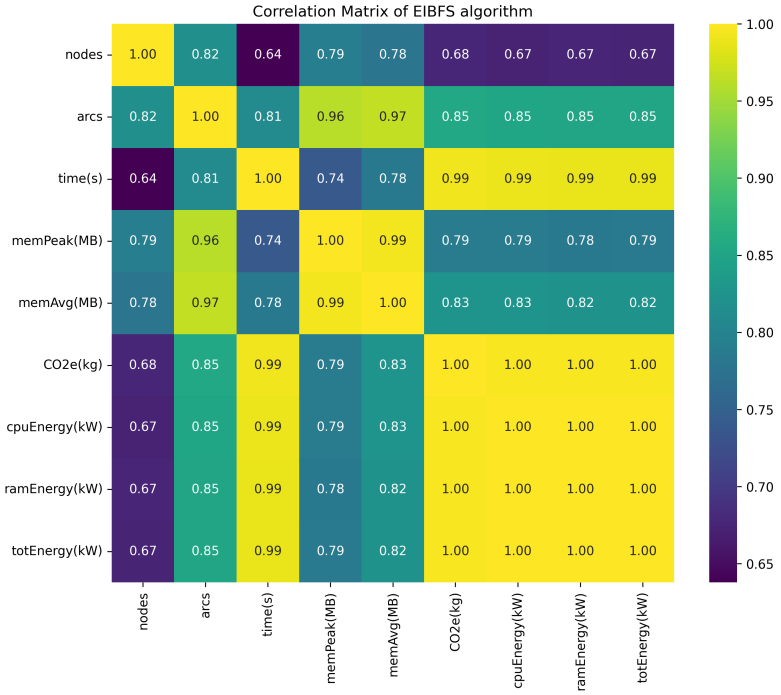


Figure 4.9: Correlation matrix for EIBFS

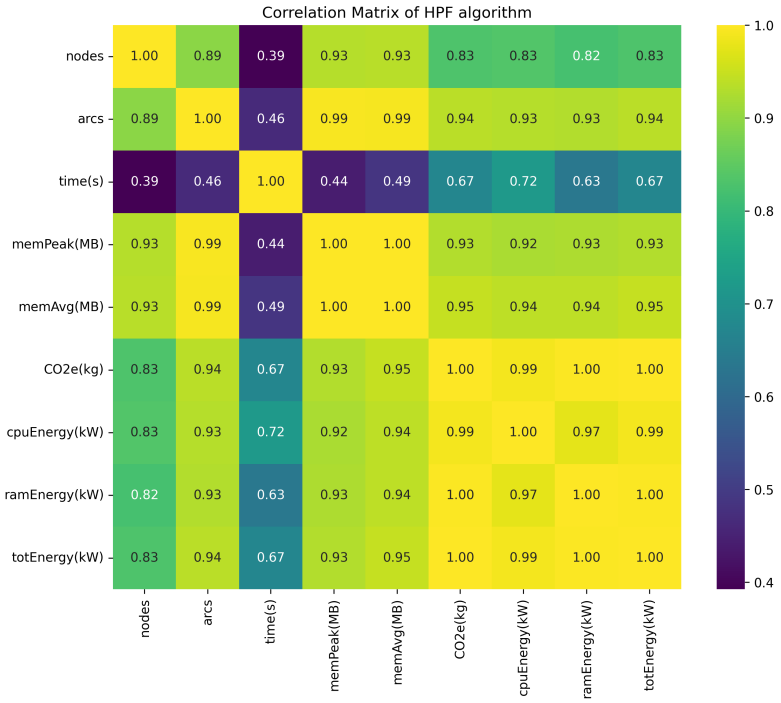


Figure 4.10: Correlation matrix for BK

and EIBFS behave when constraints on runtime are introduced, particularly when minimizing CO_2 emissions.

HPF, on the other hand, demonstrates a slightly different behavior. From Figure 4.10, we can see that the strongest correlation with CO_2 emissions is memory usage, specifically peak memory usage (memPeak) and average memory usage (memAvg), both exceeding 0.93. This suggests that rather than runtime alone, memory allocation plays a significant role in determining the carbon footprint of HPF. For this reason, in the experiments on HPF we impose constraints on peak memory usage rather than runtime.

Performance Evaluation of Min-Cut/Max-Flow Algorithms with HADA

In this section, we present the experimental results of optimizing the execution of Min-Cut/Max-Flow algorithms using HADA. The key objective was to evaluate the impact of different hardware platforms (PC, mbp19, and Leonardo) on execution time, CO_2 emissions, and memory usage.

Optimization for CO_2 Emissions Minimization Tables 4.7, 4.8, and 4.9 summarize the results obtained when minimizing CO_2 emissions. Across all three algorithms, we observe a strong preference for smaller-scale hardware platforms, with PC and mbp19 being consistently selected over Leonardo.

Inputs & Bound			Solution			
Nodes	Arcs	time(s)	Impl.	HW	CO2e	time(s)
40	400	0.1	MBK2	pc	4.38×10^{-7}	3.81×10^{-5}
120	2k	0.1	MBK2	pc	4.40×10^{-7}	5.10×10^{-5}
1k	5k	0.1	MBK	pc	4.40×10^{-7}	4.09×10^{-4}
10k	62k	0.1	BK	pc	4.40×10^{-7}	2.63×10^{-3}
44k	784k	5	MBK2	pc	6.46×10^{-6}	1.39
185k	5M	5	MBK2	pc	2.16×10^{-6}	0.28
18M	93M	10	MBK2	pc	4.69×10^{-5}	9.79
143M	1B	10	MBK2	pc	4.69×10^{-5}	9.79

Table 4.7: Experimental results for the BK algorithm

Similar trends were observed for EIBFS and HPF, as shown in Tables 4.8 and 4.9. When CO_2 emissions were minimized, the computationally lighter

Inputs & Bound			Solution			
Nodes	Arcs	time(s)	Impl.	HW	CO2e	time(s)
40	400	0.1	EIBFS new	pc	4.37×10^{-7}	3.75×10^{-5}
120	2k	0.1	EIBFS new	pc	4.40×10^{-7}	1.80×10^{-4}
1k	5k	0.1	EIBFS new	pc	4.40×10^{-7}	2.03×10^{-4}
10k	62k	0.1	EIBFS old	pc	4.40×10^{-7}	2.89×10^{-3}
44k	784k	0.5	EIBFS new	pc	8.7×10^{-7}	0.27
185k	5M	0.5	EIBFS new2	pc	1.30×10^{-6}	0.12
18M	93M	5	EIBFS new	pc	1.98×10^{-5}	3.51
143M	1B	10	none	none	none	none

Table 4.8: Experimental results for the EIBFS algorithm

Inputs & Bound			Solution			
Nodes	Arcs	memPeak(MB)	Impl.	HW	CO2e	memPeak(MB)
2.5k	50k	10	HPF-H-F	mbp19	2.11×10^{-6}	0.48
10k	75k	10	HPF-H-F	mbp19	4.91×10^{-7}	0.46
100k	300k	100	HPF-L-L	mbp19	9.56×10^{-7}	15.46
300k	500k	100	HPF-L-L	mbp19	1.46×10^{-6}	35.45
1M	5M	100	none	none	none	none
1M	5M	400	HPF-L-L	mpb19	8.25×10^{-6}	390.83
10M	25M	none	none	none	none	none
10M	25M	3000	HPF-H-L	mbp19	6.85×10^{-5}	2122.97
125M	300M	4000	HPF-L-F	mbp19	9.08×10^{-3}	3785.29

Table 4.9: Experimental results for the EIBFS algorithm

hardware architectures were consistently preferred. This result can be attributed to the following factors:

- Smaller architectures tend to have lower power consumption per computation unit, making them more efficient in terms of emissions when processing relatively small to mid-sized problem instances.
- High-performance computing (HPC) systems like Leonardo introduce computational overhead, which increases overall energy consumption, especially when handling smaller problem instances.
- The serial nature of the tested algorithm implementations does not exploit the parallelism of the HPC architecture, leading to underutilization of available resources.

Optimization for Runtime Minimization To explore whether HPC would provide a significant advantage in execution time, we re-ran HADA optimization while minimizing runtime instead of emissions. The results for BK, EIBFS, and HPF are presented in Tables 4.10, 4.11, and 4.12.

Inputs & Bound			Solution			
Nodes	Arcs	CO2e(kg)	Impl.	HW	CO2e	time(s)
40	400	0.1	MBK2	pc	4.38×10^{-7}	3.81×10^{-5}
120	2k	0.1	MBK2	pc	4.40×10^{-7}	5.10×10^{-5}
1k	5k	0.1	MBK	leonardo	4.40×10^{-7}	1.10×10^{-4}
10k	62k	0.1	MBK	leonardo	4.40×10^{-7}	2.32×10^{-3}
16k	111k	0.1	MBK	leonardo	4.44×10^{-5}	1.37
44k	784k	0.1	MBK2	leonardo	2.63×10^{-6}	0.88
185k	5M	5	MBK2	pc	2.16×10^{-6}	0.28
18M	93M	10	MBK2	pc	4.69×10^{-5}	9.79

Table 4.10: Experimental results for the BK algorithm

For smaller problem instances, we see that PC and mbp19 remain preferable (see 4.10). However, in some cases involving larger problem sizes (e.g., 16k nodes, 111k arcs), HADA selects Leonardo. Despite this, the improvement in runtime is not substantial, with the execution time on PC being approximately 4.09×10^{-4} seconds compared to 1.10×10^{-4} seconds on Leonardo.

Inputs & Bound			Solution			
Nodes	Arcs	time(s)	Impl.	HW	CO2e	time(s)
40	400	0.1	EIBFS new2	leonardo	3.01×10^{-6}	1.37×10^{-5}
120	2k	0.1	EIBFS new2	leonardo	3.01×10^{-6}	7.22×10^{-5}
1k	5k	0.1	EIBFS new2	leonardo	4.40×10^{-6}	7.23×10^{-4}
10k	62k	0.1	EIBFS new2	leonardo	4.40×10^{-6}	1.72×10^{-3}
44k	784k	0.1	EIBFS new2	leonardo	9.27×10^{-6}	0.02
185k	5M	0.1	EIBFS new2	leonardo	8.80×10^{-6}	0.09
18M	93M	5	EIBFS new	pc	1.98×10^{-5}	3.51
143M	1B	10	none	none	none	none

Table 4.11: Experimental results for the EIBFS algorithm

For EIBFS (see 4.11), results still confirm this trend even if in these case we have more configurations with leonardo.

Inputs & Bound			Solution			
Nodes	Arcs	CO2e(kg)	Impl.	HW	CO2e	time(s)
2.5k	50k	0.1	HPF-H-F	leonardo	3.58×10^{-6}	0.001
10k	75k	0.1	HPF-H-L	leonardo	2.98×10^{-6}	0.002
16k	111k	0.1	HPF-H-F	leonardo	1.02×10^{-3}	11.648
15k	200k	0.1	HPF-H-F	leonardo	1.55×10^{-3}	2.214
134k	921k	0.1	HPF-L-F	leonardo	1.46×10^{-6}	9.278
1M	5M	0.1	HPF-L-L	mpb19	8.25×10^{-6}	0.062
10M	25M	3000	HPF-H-L	mbp19	6.85×10^{-5}	5.31
125M	300M	4000	HPF-L-F	mbp19	9.08×10^{-3}	27.74

Table 4.12: Experimental results for the EIBFS algorithm

For HPF, similar results hold, with mbp19 being preferred for smaller instances, while Leonardo is selected for medium-sized problems.

Discussion: Why Does HPC Not Show a Significant Runtime Advantage? From the results obtained, we observe that even when optimizing for runtime, HADA often does not favor HPC. Several factors contribute to this:

- Sequential Algorithm Implementations: the versions of BK, EIBFS, and HPF tested in this study were single-threaded, meaning they do not take advantage of parallel processing capabilities offered by HPC clusters. As highlighted by Jensen et al. [16], specialized parallel implementations of max-flow algorithms exist, and these would likely benefit more from HPC architectures.

- Computational Overhead in HPC systems: HPC environments like Leonardo introduce additional computational overhead due to factors such as job scheduling, data transfer, and process management. For small and medium-sized instances, this overhead can negate the potential runtime benefits, leading to comparable or even worse performance compared to commodity hardware.

Chapter 5

HADA-as-a-Service

5.1 HADA Web Application

The HADA framework, which was used to conduct the experiments in this study, has also been implemented as a web application. This implementation provides a user-friendly interface for defining optimization problems, specifying constraints, and selecting the appropriate computational setup. The interface of the HADA web service application is depicted in 5.1.

The web interface is designed to simplify the optimization process by offering several key functionalities:

- Algorithm Selection: A dropdown menu allows the user to select the target algorithm for optimization.
- Input File Uploader: For input-dependent algorithms, users can upload input datasets required for execution.
- Geographic Selection: The user can select the country where the computation will be performed. This is crucial as it affects the carbon emissions estimation based on region-specific Carbon Intensity values.
- Optimization Target: The user can specify which metric to optimize, such as execution time, energy consumption, or carbon emissions.

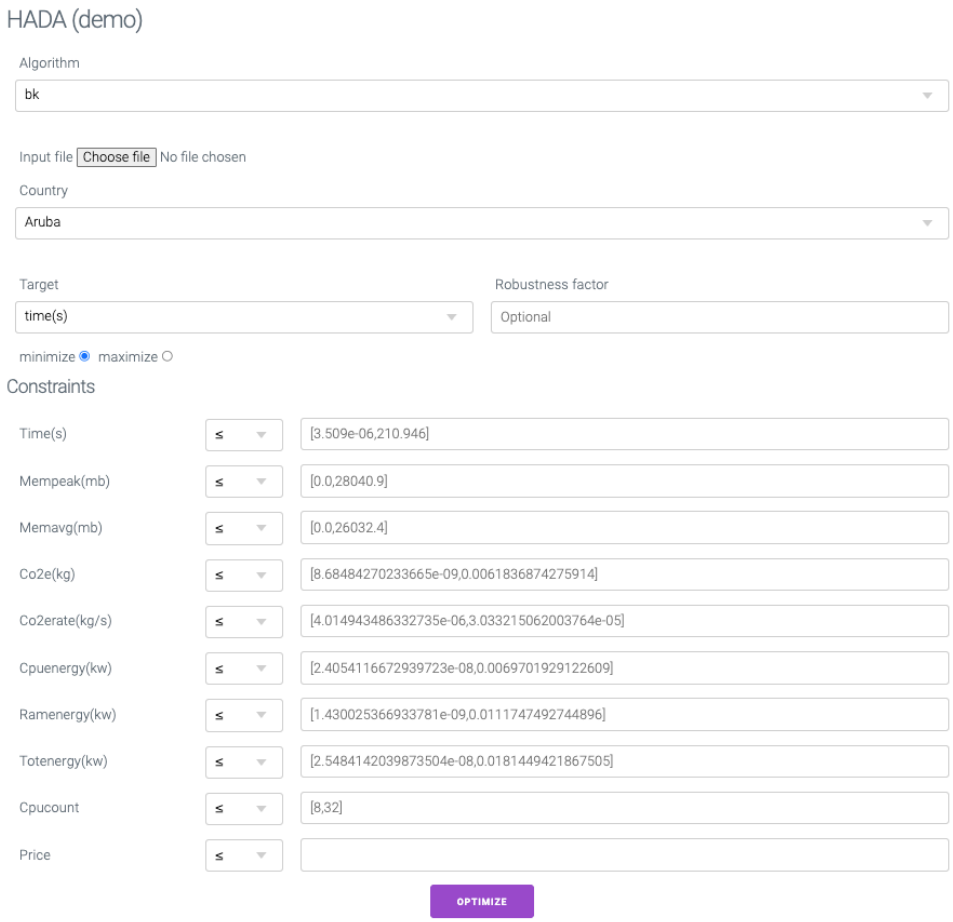


Figure 5.1: The UI for the HADA web service application

- Robustness Factor: An optional setting that allows to specify confidence level for the user defined constraints
- Constraint Specification: Users can define constraints on various performance metrics (e.g., execution time, memory usage, CO_2 emissions). Constraints can be defined as:
  - $\leq$ (Upper bound)
  - $\geq$ (Lower bound)
  - $=$ (Fixed value)
- The ”Optimise” button, which launches the optimization process.

When the optimization starts, the HADA backend performs the following

steps:

- Model Checking: The system verifies if pre-trained models exist for the required combination of (Algorithm, Hardware, Target). If a model is unavailable, HADA trains a new one.
- Machine Learning Models: The ML models used are Decision Tree Regressors with a maximum depth of 10. These models were found to provide a good balance between prediction accuracy and computational complexity [6].
- Integration with Optimization Engine: The trained ML models are incorporated into the optimization problem as constraints.
- Solving the Optimization Model: The mathematical model, including user-defined constraints, is passed to IBM CPLEX, which attempts to find an optimal solution.

An example of an optimization request sent through the HADA GUI is depicted in 5.2.

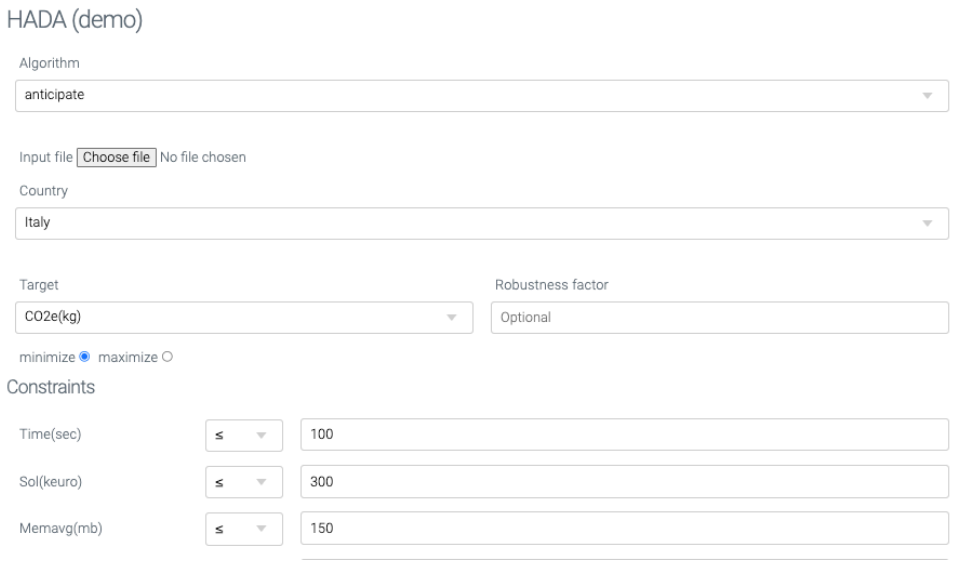


Figure 5.2: Example of an optimisation request sent to HADA via the GUI

If a feasible solution is found, the HADA GUI displays the optimal configuration, including the selected hardware platform, algorithm hyperparameters, and the estimated performance metrics, as shown in Figure 5.3.

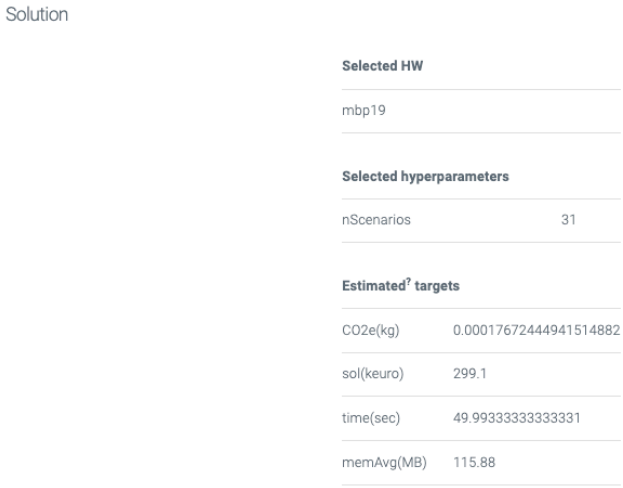


Figure 5.3: The solution provided for the above example in Figure 5.2

One of the key aspects of this work was to extend HADA account for geographical variations in Carbon Intensity when computing emissions. To demonstrate this, we perform the same optimization request but change the execution location. For example, Figure 5.4 shows the results when selecting Turkmenistan as the execution country. According to Our World in Data [8], Turkmenistan had the highest Carbon Intensity in 2023, with 1.168 kg of CO_2 per kWh.

Comparing this to Figure 5.3, where Italy was the selected country, we observe a significant increase in CO□ emissions:

- Italy: $\approx 1.77 \times 10^{-4}$ kg CO_2e
- Turkmenistan: $\approx 6.98 \times 10^{-4}$ kg CO_2e

This increase highlights the critical role of geographic selection when running large-scale computations. While the absolute difference in emissions for a single execution might seem small, the impact becomes substantial when

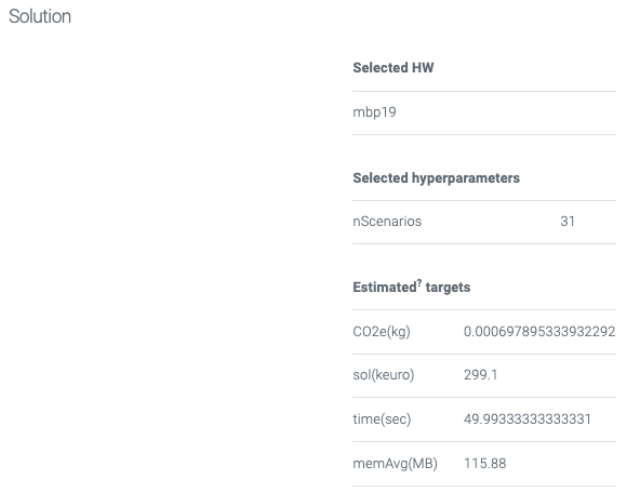


Figure 5.4: Solution provided for the same constraint as in 5.2, but this time selecting ”Turkmenistan” as ”Country”

considering large workloads and multiple executions. This reinforces the importance of choosing greener energy grids whenever possible.

5.1.1 Integration of Benchmark Data into HADA

To expand HADA with additional benchmark data, datasets must be structured and formatted correctly. The standard organization for datasets is shown in Figure 5.5.

Dataset Format

HADA expects datasets in `.csv` format, with files organized by **algorithm** and **platform**. Each dataset contains:

- Algorithm Name
- Hardware Platform
- Hyperparameters (e.g., number of scenarios for ANTICIPATE)
- Target Performance Metrics (e.g., execution time, energy consumption, CO_2 emissions)

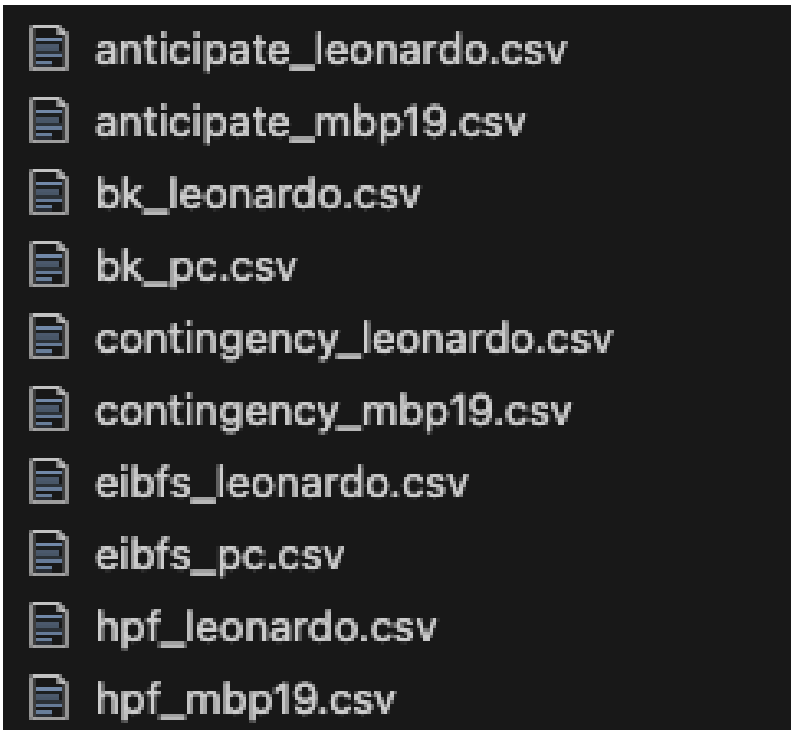


Figure 5.5: Organization of benchmark datasets for integration into HADA.

- Optional Input Parameters for input-dependent algorithms

Metadata JSON Configuration

In addition to dataset files, HADA requires a **JSON configuration file** specifying:

- Algorithm details
- Hardware platform information
- Hyperparameter definitions
- Target performance metrics
- Constraints (if any)

Below is an example JSON structure:

```
{
```

```
    "name": "anticipate",
    "HW_ID": "macbook",
    "hyperparams": [
        {"ID": "num_scenarios", "type": "int", "LB": 1, "UB": 100}
    ],
    "targets": [
        {"ID": "time", "LB": null, "UB": null},
        {"ID": "memory", "LB": null, "UB": null},
        {"ID": "emissions", "LB": null, "UB": null}
    ]
}
```

This approach allows HADA to seamlessly integrate new datasets, expanding its optimization capabilities to additional algorithms and hardware platforms.

Chapter 6

Conclusions

This work extends HADA by integrating carbon emission constraints, enhancing its applicability for sustainable AI hardware selection. Through experimental benchmarks on laptops and HPC systems, we validated the framework's ability to balance performance and environmental impact. The web-based prototype enables users to make informed decisions when configuring AI workloads under sustainability constraints.

From the results obtained in both ANTICIPATE and CONTINGENCY experiments, it is evident that platform selection plays a critical role in emissions optimization. Higher performance platforms like leonardo could be beneficial when the superior computational power brings significant improvements in the algorithm runtime. Usage of HPC architectures like leonardo could be convenient to perform heavy parallel compuations, expecially with runtime constraints, or when the problem size requires more memory. For light or moderate workloads, with little to no parallelization, expecially when energy efficiency is the main focus, one should stick with smaller architectures.

Bibliography

[1] N. Bannour et al. “Evaluating the Carbon Footprint of NLP Methods: A Survey and Analysis of Existing Tools”. In: *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*. Virtual: Association for Computational Linguistics, 2021, pp. 11–21. DOI: 10.18653/v1/2021.sustainlp-1.2. URL: https://aclanthology.org/2021.sustainlp-1.2/.

[2] Y. Boykov and V. Kolmogorov. “An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.9 (2004), pp. 1124–1137. DOI: 10.1109/TPAMI.2004.60.

[3] B. Courty et al. *mlco2/codecarbon: v2.4.1*. Version v2.4.1. 2024. DOI: 10.5281/zenodo.11171501. URL: https://doi.org/10.5281/zenodo.11171501.

[4] A. De Filippo, M. Lombardi, and M. Milano. “How to Tame Your Anticipatory Algorithm”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 1071–1077. DOI: 10.24963/ijcai.2019/150. URL: https://doi.org/10.24963/ijcai.2019/150.

[5] A. De Filippo, M. Lombardi, and M. Milano. “Off-Line and On-Line Optimization Under Uncertainty: A Case Study on Energy Management”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings*. Delft, The Netherlands: Springer-Verlag, 2018, pp. 100–116. ISBN: 978-3-319-93030-5. DOI: 10.1007/978-3-319-93031-2_8. URL: https://doi.org/10.1007/978-3-319-93031-2_8.

[6] A. De Filippo et al. “HADA: An automated tool for hardware dimensioning of AI applications”. In: *Knowledge-Based Systems* 251 (2022), p. 109199. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2022.109199. URL: https://www.sciencedirect.com/science/article/pii/S0950705122005974.

[7] J. Dodge et al. “Measuring the Carbon Intensity of AI in Cloud Instances”. In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. FAccT ’22. Seoul, Republic of Korea: Association for Computing Machinery, 2022, pp. 1877–1894. ISBN: 9781450393522. URL: https://doi.org/10.1145/3531146.3533234.

[8] Ember, E. Institute, and O. W. in Data. *Carbon Intensity of Electricity Generation – Ember and Energy Institute*. Dataset. Major processing by Our World in Data. Based on Ember’s ”Yearly Electricity Data” and Energy Institute’s ”Statistical Review of World Energy”. 2024. URL: https://ourworldindata.org/grapher/carbon-intensity-electricity (visited on 03/06/2025).

[9] A. Faiz et al. *LLMCarbon: Modeling the end-to-end Carbon Footprint of Large Language Models*. 2024. arXiv: 2309.14393 [cs.CL]. URL: https://arxiv.org/abs/2309.14393.

[10] L. R. Ford and D. R. Fulkerson. “Maximal Flow Through a Network”. In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

[11] A. V. Goldberg et al. “Faster and more dynamic maximum flow by incremental breadth-first search”. In: *Algorithms-ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*. Springer. 2015, pp. 619–630.

[12] U. Gupta et al. “Chasing Carbon: The Elusive Environmental Footprint of Computing”. In: *CoRR* abs/2011.02839 (2020). arXiv: 2011.02839. URL: https://arxiv.org/abs/2011.02839.

[13] P. Henderson et al. “Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning”. In: *Journal of Machine Learning Research* 21.248 (2020), pp. 1–43. URL: http://jmlr.org/papers/v21/20-312.html.

[14] D. S. Hochbaum. “The pseudoflow algorithm: A new algorithm for the maximum-flow problem”. In: *Operations research* 56.4 (2008), pp. 992–1009.

[15] E. J. Husom et al. *The Price of Prompting: Profiling Energy Use in Large Language Models Inference*. 2024. arXiv: 2407.16893 [cs.CY]. URL: https://arxiv.org/abs/2407.16893.

[16] P. M. Jensen et al. “Review of Serial and Parallel Min-Cut/Max-Flow Algorithms for Computer Vision”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.2 (2023), pp. 2310–2329. DOI: 10.1109/TPAMI.2022.3170096.

[17] A. Lacoste et al. “Quantifying the Carbon Emissions of Machine Learning”. In: *arXiv preprint* (2019). arXiv: 1910.09700. URL: https://arxiv.org/abs/1910.09700.

[18] L. Lannelongue, J. Grealey, and M. Inouye. "Green Algorithms: Quantifying the Carbon Footprint of Computation". In: *Advanced Science* 8.12 (2021), p. 2100707. DOI: https://doi.org/10.1002/advs.202100707. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/advs.202100707. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/advs.202100707.

[19] M. Lombardi, M. Milano, and A. Bartolini. "Empirical decision model learning". In: *Artificial Intelligence* 244 (2017). Combining Constraint Solving with Mining and Learning, pp. 343–367. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2016.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0004370216000126.

[20] N. Maslej et al. *The AI Index 2024 Annual Report*. Tech. rep. AI Index Steering Committee, Stanford University, 2024.

[21] D. A. Patterson et al. "Carbon Emissions and Large Neural Network Training". In: *CoRR* abs/2104.10350 (2021). arXiv: 2104.10350. URL: https://arxiv.org/abs/2104.10350.

[22] S. Samsi et al. *From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference*. 2023. arXiv: 2310.03003 [cs.CL]. URL: https://arxiv.org/abs/2310.03003.

[23] R. Schwartz et al. "Green AI". In: *CoRR* abs/1907.10597 (2019). arXiv: 1907.10597. URL: http://arxiv.org/abs/1907.10597.

[24] J. Sevilla and E. Roldán. *Training Compute of Frontier AI Models Grows by 4-5x per Year*. Accessed: 2025-03-03. 2024. URL: https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year.

[25] E. Strubell, A. Ganesh, and A. McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *CoRR* abs/1906.02243 (2019). arXiv: 1906.02243. URL: http://arxiv.org/abs/1906.02243.

[26] N. Thompson et al. “Deep Learning’s Diminishing Returns: The Cost of Improvement is Becoming Unsustainable”. In: *IEEE Spectrum* 58 (2021), pp. 50–55. ISSN: 0018-9235.

[27] H. Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971.

[28] C. Wu et al. “Sustainable AI: Environmental Implications, Challenges and Opportunities”. In: *CoRR* abs/2111.00364 (2021). arXiv: 2111.00364. URL: https://arxiv.org/abs/2111.00364.

Acknowledgements

I’m very grateful to the inventor of the Prolog language, without whom this thesis couldn’t exist. I’d also like to acknowledge my advisor Prof. Mario Rossi by tail-recursively acknowledging my advisor.