https://doi.org/10.15388/vu.thesis.377
https://orcid.org/0000-0002-6178-5492

VILNIUS UNIVERSITY

Vaidas Jusevičius

Research and development of an open-source algebraic modeling and mathematical optimization system

DOCTORAL DISSERTATION

Natural Sciences,
Informatics (N 009)

Vilnius 2022

This dissertation was written between 2017 and 2021 at Vilnius University.

Academic supervisor:

Prof. Dr. Remigijus Paulavičius (Vilnius University, Natural Sciences, Informatics – N 009).

Defence Panel:

Chairwoman – Prof. Dr. Olga Kurasova (Vilnius University, Natural Sciences, Informatics – N 009).

Members:

Prof. Dr. Leocadio González Casado (University of Almeria, Spain, Natural Sciences, Informatics – N 009),

Assoc. Prof. Dr. Algirdas Lančinskas (Vilnius University, Natural Sciences, Informatics – N 009),

Dr. Viktor Medvedev (Vilnius University, Natural Sciences, Informatics – N 009),

Prof. Dr. Dmitrij Šešok (Vilnius Gediminas Technical University, Technological Sciences, Informatics Engineering – T 007).

The dissertation shall be defended at a public meeting of the Dissertation Defence Panel at 12 p.m. on the 28th of September, 2022 in Room 203 of the Institute of Data Science and Digital Technologies of Vilnius University. Address: Akademijos street 4, LT-08412, Vilnius, Lithuania, Tel. +37052109300 ; e-mail: info@mii.vu.lt.

The text of this dissertation can be accessed at the library of Vilnius University, as well as on the website of Vilnius University:
www.vu.lt/lt/naujienos/ivykiu-kalendorius

https://doi.org/10.15388/vu.thesis.377
https://orcid.org/0000-0002-6178-5492

VILNIAUS UNIVERSITETAS

Vaidas Jusevičius

Atvirojo kodo algebrinio modeliavimo ir matematinio optimizavimo sistemos kūrimas ir tyrimas

DAKTARO DISERTACIJA

Gamtos mokslai,
Informatika (N 009)

Vilnius 2022

Disertacija rengta 2017 – 2021 metais Vilniaus universitete.

Mokslinis vadovas:
prof. dr. Remigijus Paulavičius (Vilniaus universitetas, gamtos mokslai, informatika – N 009).

Gynimo taryba:
Pirmininkė – **prof. dr. Olga Kurasova** (Vilniaus universitetas, gamtos mokslai, informatika – N 009).
Nariai:
prof. dr. Leocadio González Casad (Almerijos universitetas, Ispanija, gamtos mokslai, informatika – N 009),
doc. dr. Algirdas Lančinskas (Vilniaus universitetas, gamtos mokslai, informatika – N 009),
dr. Viktor Medvedev (Vilniaus Universitetas, gamtos mokslai, informatika – N 009),
prof. dr. Dmitrij Šešok (Vilniaus Gedimino technikos universitetas, technologijos mokslai, informatikos inžinerija – T 007).

Disertacija ginama viešame Gynimo tarybos posėdyje 2022 m. rugsėjo mėn. 28 d. 12 val. Vilniaus universiteto Duomenų mokslo ir skaitmeninių technologijų instituto 203 auditorijoje. Adresas: Akademijos g. 4, LT-08412, Vilnius, Lietuva, tel. +37052109300 ; el. paštas info@mii.vu.lt.

Disertaciją galima peržiūrėti Vilniaus univeristeto bibliotekoje ir VU interneto svetainėje adresu:
www.vu.lt/lt/naujienos/ivykiu-kalendorius

ABSTRACT

In this dissertation, the concept of a universal mathematical optimization system consisting of an algebraic modeling language and an open source tool capable of building an optimization model and solving it with potentially multiple underlying optimization solvers is presented.

First, the main principles of algebraic modeling are presented, the essential characteristics of modern algebraic modeling languages (AMLs) are reviewed, and the most prominent AMLs are identified. Later, a bibliometric analysis of the research field is conducted by analyzing publications from the year 2000 to the present day. Such bibliometric analysis allows to confirm the importance of the research topic and validates the correctness of the choice for the most prominent AMLs.

Next, an extensive theoretical and experimental analysis of the characteristics of four of the most prominent algebraic modeling languages (`AMPL`, `GAMS`, `JuMP`, and `Pyomo`) and the modeling systems supporting them is performed. In the theoretical comparison, it is evaluated how the reviewed modern AMLs match the requirements for modern AMLs identified earlier. A purpose-built test model library is used to perform extensive benchmarks in experimental analysis. Then the best performing algebraic modeling languages are determined by comparing the time needed to create model instances for a specific optimization problem and analyze the impact that the presolve procedures performed by various algebraic modeling languages have on the actual problem-solving times. Insights on which algebraic modeling languages perform the best and the features that we consider essential in the current mathematical optimization landscape are provided.

Afterwards, the main gaps within the existing algebraic modeling languages and tools are distilled. Such gaps include varying performance, limited cross-compatibility, complex syntax, and different support for solvers, advanced features, and problem types.

Later, a concept of a state-of-the-art universal optimization system for algebraic modeling languages and mathematical optimization is proposed. Using the system does not require specific algebraic language knowledge, allows for solving problems using different solvers, and utilizes the best

characteristics of existing algebraic modeling languages.

To assess the feasibility of the proposal, a prototype of such a web-based tool using React.js and Java technological stack is implemented. Later, the prototype is used to compare its characteristics with other AMLs and provide an overview of how it addresses the gaps identified earlier.

Lastly, clear extension points and ideas on how such a tool could be further improved are provided. This includes increasing the expression power of the modeling language, improving the user interface, and including new features, such as presolving or parallel model creation.

SANTRAUKA

Šioje disertacijoje yra pristatoma universalios matematinio optimizavimo sistemos koncepcija, sudaryta iš algebrinės modeliavimo kalbos ir atvirojo kodo įrankio gebančio sukonstruoti sprendžiamos problemos modelį bei vėliau jį išspręsti pasinaudojant kitomis algebrinio modeliavimo sistemomis ir jų palaikomais optimizavimo įrankiais (sprendėjais).

Visų pirma, pristatomi pagrindiniai algebrinio modeliavimo principai, apžvelgiamos svarbiausios algebrinių modeliavimo kalbų savybės bei identifikuojamos svarbiausios algebrinės modeliavimo kalbos. Toliau yra pateikiama bibliometrinė tematikos literatūros apžvalga apimanti publikacijas nuo 2000-ųjų metų. Bibliometrinė analizė leidžia patvirtinti temos aktualumą ir svarbiausių algebrinių modeliavimo kalbų parinkimo korektiškumą.

Tiriamoji darbo dalis pradedama išsamia teorine ir praktine identifikuotų svarbiausių algebrinio modeliavimo kalbų (`AMPL`, `GAMS`, `JuMP` ir `Pyomo`) ir jas palaikančių sistemų analize. Teorinėje analizėje įvertinama kaip apžvelgiamos kalbos atitinka modernioms algebrinio modeliavimo kalboms keliamus reikalavimus. Eksperimentinėje dalyje yra sukonstruojama testinė optimizavimo uždavinių biblioteka, kurios pagalba yra atliekami išsamūs nagrinėjamų sistemų našumo tyrimai. Tai leidžia identifikuoti našiausias algebrines modeliavimo kalbas vertinant pagal modelio egzemplioriaus kūrimo laiką konkrečioms uždavinių grupėms. Taip pat, nustatomas potencialus uždavinio sprendimo pagreitėjimas naudojant išankstinio sprendimo algoritmus pateikiamus kai kurių algebrinio modeliavimo sistemų. Apibendrinant tyrimus yra pateikiamos įžvalgos, nurodant kurios iš algebrinių kalbų bei sistemų pasirodė pranašiausios ir patvirtinamos kertinės šiuolaikinės algebrinio modeliavimo kalbos savybės.

Toliau darbe yra identifikuojami pagrindiniai trūkumai, kuriais pasižymėti nagrinėtos algebrinio modeliavimo kalbos ir įrankiai. Iš kurių galima išskirti tokius, kaip varijuojantis našumas, ribotas tarpusavio suderinamumas, sudėtinga sintaksė, skirtingas optimizavimo sprendėjų, uždavinių tipų, bei papildomų funkcijų palaikymas.

Atsižvelgiant į identifikuotus trūkumus yra pasiūlomas universalios optimizavimo sistemos konceptas, apimantis algebrines modeliavimo kalbas

ir matematinį optimizavimą. Siūloma sistema nereikalauja specifinių algebrinių modeliavimo kalbų žinių, leidžia spręsti optimizavimo uždavinius skirtingų sprendėjų pagalba ir stengiasi išnaudoti geriausias egzistuojančių algebrinių modeliavimo kalbų ir sistemų savybes.

Siekiant įsitikinti šio koncepto įgyvendinamumu, yra realizuojamas pasiūlytos koncepcijos prototipas žiniatinklio aplinkoje. Prototipui realizuoti pasirenkamos React.js bei Java technologijos. Vėliau, pasinaudojant prototipu, yra atliekamas universalios optimizavimo sistemos ir egzistuojančių algebrinių modeliavimo sistemų savybių palyginimas.

Galiausiai, yra nurodomi prototipo plėtimo taškai ir idėjos, kaip pastarasis gali būti vystomas ateityje. Tai algebrinio modeliavimo kalbos išraiškos galios plėtimas, vartotojo sąsajos patobulinimai, savybių, tokių kaip išankstinis sprendimas ar lygiagretus modelio užkrovimas įgyvendinimas.

CONTENTS

INTRODUCTION

Research Context And Motivation

Many real-world problems are routinely solved using modern optimization tools [e.g., 1, 27, 34, 65, 62, 64]. Internally, these tools use the combination of a mathematical model with an appropriate solution algorithm [e.g., 14, 21, 35, 49, 65, 63, 62, 70, 71] to solve the problem at hand. Thus, the way mathematical models are formulated is critical to the impact of optimization in real life. Examples of real-life problems include production and shipment by firms, investment planning, macroeconomics stabilization, water distribution networks, oil refineries, petrochemical plants, applied general equilibrium, international trade of aluminum and copper, and many more (see, e.g., [29]).

Mathematical modeling is the process of translating real-world business problems into mathematical formulations whose theoretical and numerical analysis can provide insight, answers, and guidance beneficial for the originating application [44], including the current Covid-19 pandemic [68]. Algebraic modeling languages (AMLs) are declarative optimization modeling languages, which bridge the gap between model formulation and the proper solution technique [27]. They enable the formulation of a mathematical model as a human-readable set of equations without requiring to specify how the described model should be solved or what specific solver should be used.

Models written in an AML are known for a high degree of similarity to the mathematical formulation. This aspect distinguishes AMLs from other types of modeling languages, like object-oriented (e.g., `OptimJ` [5]), solver specific (e.g., `LINGO` [50]), or general-purpose (e.g., `TOMLAB` [73]) modeling languages. Such an algebraic design approach allows practitioners without specific programming or modeling knowledge to be efficient in describing the problems to be solved. It is also important to note that AML is then responsible for creating a problem instance that a solution algorithm can tackle [44]. Since many AMLs are integral parts of a specific modeling system, it is essential to isolate the responsibilities of a modeling language from the overall system.

In general, AMLs are sophisticated software packages that provide a crucial link between an optimization model's mathematical concept and the complex algorithmic routines that compute optimal solutions. Typically, AML software automatically reads a model and data, generates an instance, and conveys it to a solver in the required form [25].

From the late 1970s, many AMLs were created (e.g., `GAMS` [55], `AMPL` [24]) and are still actively developed and used today. Lately, new open-source competitors to traditional AMLs have started to emerge (e.g., `Pyomo` [37, 38], `JuMP` [18, 54]). Therefore, a review and comparison of the traditional and emerging AMLs are needed to examine how the current landscape of AMLs looks.

Until now, some comparisons of AMLs were made based on questionnaires sent out by the vendors [26]. However, there is a lack of extensive theoretical and experimental analysis around the characteristics of the most prominent AMLs (`AMPL`, `GAMS`, `JuMP`, and `Pyomo`) and the modeling systems supporting them.

Thus, there is a need to continue research by further distilling the main gaps within the existing AMLs and the optimization systems supporting them. This enabling to identify the requirements for the concept of more universal optimization systems combining the best characteristics of existing AMLs. Work in this direction has already been started with suggestions such as using LaTeX as a foundation for the AML tool by Triantafyllidis and Papageorgiou [74], or CasADi [3] an open-source tool for nonlinear optimization and algorithmic differentiation. However, another, more extensible, and user-friendly alternative can be provided, which would benefit not only practitioners with a mathematical background but also those just starting their path of learning mathematical optimization.

The object of the Thesis

The object of the thesis is software systems supporting algebraic modeling languages for solving real-world optimization problems. Both commercial and open-source algebraic modeling software packages are in scope.

Aims and Tasks of the Research

The aim of the thesis is to propose a concept for the universal optimization system including algebraic modeling language and modeling system combining the best characteristics of the most prominent AMLs.

In order to achieve the aim of the thesis, the following research tasks must be accomplished:

1. Identify the main features of modern algebraic modeling languages and choose the most prominent AML currently existing on the market and matching the criteria.
2. Theoretically review chosen algebraic modeling systems, compare their differences, and identify each of their shortcomings.
3. Create a library of optimization problems for conducting performance tests of the chosen AMLs.
4. Perform an experimental analysis of chosen AMLs by running performance tests of the created testing library.
5. Ensure reproducibility of the experimental analysis.
6. Based on theoretical and experimental analysis decide and define areas where existing algebraic modeling systems can be improved.
7. Propose the concept of a universal optimization system that includes a generic algebraic modeling language and an open-source system supporting it.
8. Provide the prototype of the proposed universal optimization system proving its viability and providing a foundation for future development of a fully-featured universal optimization system.

Research Methodology

To analyze the scientific results received in the fields of algebraic modeling and mathematical optimization, information retrieval, organization, analysis,

comparative analysis, and generalization methods have been used. For the interpretation of the experimental investigation, statistical analysis was applied to evaluate the efficiency of algebraic modeling languages.

Scientific Novelty of the Work

This dissertation was written in a field of the doctoral study program of Informatics. However, some of the research required to support defended statements places the work within an intersection of doctoral study programs of Informatics and Informatics Engineering. While within the field of Informatics, an in-depth comparison of algebraic modeling languages was made, it was required to create testing library and tooling to support it so entering the field of Informatics Engineering. Also, while within the field of Informatics a proposal for universal optimization system was made, it was required to assess the viability of the proposal by implementing the prototype system thus once more crossing to the field of Informatics Engineering.

The main novelties of this dissertation are the following:

1. A comprehensive comparative analysis of similarities and differences between the most prominent algebraic modeling languages (`AMPL`, `GAMS`, `JuMP`, and `Pyomo`) and the supporting modeling systems has been carried out. Until now, comparisons of AMLs were limited either in the depth of the characteristics assessed or in the way they were conducted (e.g., focusing on the answers provided by the vendors via questionnaires). Here, in one place, not only was the conformance to the main requirements of modern AML assessed, but also the comparison of basic and advanced features, usability, portability, and pricing was made.

2. An open library of test and practical optimization problems (algebraic models) defined in various AMLs and consumable by modern state-of-the-art solvers has been created. The essential uniqueness of this library is not even its size or the variety of model formats, but the openness, decentralization, and tooling support. Open-source and

GitHub based repository[1] provides everyone the opportunity to contribute to the growth of this library, while the scripts and tools provided by the author allow an easy way to convert between different AMLs.

3. Efficient model instance creation and model simplification have been assessed from the prism of operations research [42]. An experimental benchmark of the model instance creation time was conducted against a testing model library consisting of almost three hundred models covering different problem types and sizes. This is the largest model instance creation time benchmark to this day. It highlights significant variation in efficiency between different AMLs and challenges some of the previous benchmarks performed by the creators of AMLs. Another experimental benchmark against the testing library by using `AMPL` presolver to simplify the model assessed the presolve impact on solving and identified that the positive impact of presolving is always more significant than the negative one.

4. On the basis of comparative and experimental analysis, differences and shortcomings among the most prominent AMLs were identified. This led to the proposal for a concept of an open-source universal optimization system. It combines the best characteristics of existing algebraic modeling languages while also providing an intuitive and user-friendly optimization problem formulation (i.e. model definition) process. Two main building blocks have been defined, the WebAML language to capture problem semantics, and an optimization system acting as an orchestrator between the WebAML language and the underlying AMLs.

5. A concept of a universal optimization system having no analogs in the world was defined and a prototype of such a system was developed [43]. It does not require any specific algebraic language knowledge and allows for solving problems using different mathematical optimization solvers. It can be used by both researchers and practitioners in various sectors

[1] https://github.com/vaidasj/alg-mod-rev/tree/master/gamslib

of the economy, and the tool itself can enable faster and more efficient model development for decision-makers.

Defended Statements

1. There are quite a few powerful modeling environments and algebraic modeling languages, however, neither of them provide a complete feature set required for efficient and intuitive modeling and solving of optimization problems.

2. A proposal for the universal optimization system, combining the best characteristics of multiple prominent AMLs helps to address the shortcomings identified within the prominent AMLs.

3. Architectural decisions made in designing the prototype of a universal optimization system make it the foundation for a universal optimization toolkit. The toolkit could be extended with other state-of-the-art features such as presolving, distributed solving, or the best solver selection based on the type of model.

Approbation of the Research

The results of this research were published in the following reviewed scientific periodical publications:

1. **Jusevičius, V.** and Paulavičius, R. ”Web-Based Tool for Algebraic Modeling and Mathematical Optimization”. Mathematics, 2021, 9 (21), 2751. DOI: 10.3390/math9212751.

2. **Jusevičius, V.**, Oberdieck, R. and Paulavičius, R. ”Experimental Analysis of Algebraic Modelling Languages for Mathematical Optimization”. Informatica, 2021, 32 (2), 283–304. DOI: 10.15388/21-INFOR447.

The results of this research were presented at the following international conferences:

1. **Jusevičius, V.** and Paulavičius, R. “Experimental Analysis of Algebraic Modeling Languages For Social Behavior Modeling”, The International EURO mini Conference Modelling and Simulation of Social-Behavioural Phenomena in Creative Societies, September 18–20, 2019. Vilnius, Lithuania.

2. **Jusevičius, V.** and Paulavičius, R. ”Web-based tool for algebraic modeling languages”, EURO 2021: 31st European Conference on Operational Research, July 11-13, 2021. Athens, Greece.

Structure of the Dissertation

The dissertation consists of an introduction, 5 chapters, general conclusions, bibliography, summary in Lithuanian, and a list of publications. The total scope of the dissertation is 144 pages, including 12 figures, 26 tables, 5 equations, and 12 listings. The dissertation was based on 81 literature sources.

The introduction describes the research context, presents the problem statement, discusses the motivation, aims, objectives of the research states, research questions, describes the research methods, and approbation of the research. Chapter 1 gives theoretical background of mathematical optimization, algebraic modeling languages and modeling systems. Chapter 2 provides a comparative analysis on how the reviewed modern AMLs match the current needs of practitioners. Chapter 3 introduces the testing library of optimization problems used for the experimental analysis of AMLs and conducts an experimental analysis of the performance for chosen AMLs. Chapter 4 summarizes the theoretical and practical differences identified within AMLs. Chapter 5 discusses and proposes a concept of a universal optimization system consisting of the WebAML language and tools capable of building WebAML models and solve them using underlying AMLs. Additionally, in this chapter, a prototype of universal optimization is developed and its extensibility is discussed. And finally, at the end of the dissertation, the main results and conclusions of this thesis are summarized.

1 MATHEMATICAL OPTIMIZATION AND ALGEBRAIC MODELING LANGUAGES

When solving a mathematical optimization problem, one tries to minimize or maximize a quantity involved in the decision process, such as elapsed time or cost, with the given freedom within a set of restrictions. Optimization problems arise in almost every industry sector, for example in product and process design, manufacturing, logistics, and even strategic planning. Therefore, optimization involves finding the best solution to a given problem globally, or at least in a local neighborhood. Except for very simple cases, optimization problems cannot be solved by simulation (simulate the studied processes, evaluate the objective function and compare the results). Indeed, testing a finite number of points or scenarios can never produce sufficient knowledge proving that a solution is optimal. However, because experts in simulation engineering have developed the intuition and experience to select appropriate scenarios for evaluation, simulation software is available to perform their assessment. Simulation can lead to reasonable results, but there is no guarantee that the optimal solution or even one closest to the optimal will be found. This is especially troublesome for complex issues or those that require decisions with high financial impact [44].

To solve a real-world problem by mathematical optimization, the problem must be represented by a mathematical model, that is, a set of mathematical relations (e.g. equalities, inequalities, logical conditions) which represent an abstraction from the real-world problem. This translation is part of the modeling phase, and is by no means trivial. Mathematical models for optimization usually lead to structured problems such as linear programming (LP) [67] problems, mixed integer linear programming (MILP) problems, nonlinear programming (NLP) problems, and mixed integer nonlinear programming (MINLP) [23] problems. During the modeling phase, real-world optimization problems are structured according to the base objects, variables, objective functions, and their constraints. Then it is needed to put such a model into the machine to solve the problem [44].

Algebraic modeling languages are by far the best approach to do this.

They make it possible to define an optimization problem close to its mathematical formula, they are flexible and open to be quickly changed and adjusted. Once the model is defined, a solver, software package with an implemented algorithm capable of solving the problem, is required. In the ideal case, the optimal solution is returned. In practice, when trying to solve real-world problems, it is common to find that a problem returned with the problem statement is not feasible. Therefore, a modeling system must also support the identification of such cases [44].

In the the sub-sections below mathematical optimization problems and algebraic modeling languages are described and analyzed in more details.

1.1 Types of mathematical optimization problems

As noted earlier, an important step in the optimization process is classifying optimization model, since algorithms for solving optimization problems are tailored to a particular type of problem.

Kallrath et al. states that looking only to algebraic optimization (not allowing differential or integral relationships), the largest class of problems are mixed integer nonlinear optimization problems and formally describes them in a following way. For vectors $\mathrm{x}^{\mathrm{T}} = (x_1, \ldots, x_{n_c})$ and $\mathrm{y}^{\mathrm{T}} = (y_1, \ldots, y_{n_d})$ of n_c continuous and n_d discrete variables, the augmented vector $\mathrm{x}^{\mathrm{T}}_{\oplus} = \mathrm{x}^{\mathrm{T}} \oplus \mathrm{y}^{\mathrm{T}}$, an objective function $f(\mathrm{x}, \mathrm{y})$, n_e equality constraints h(x, y) and n_i inequalities constraints g(x, y), an optimization problem is called Mixed Integer Nonlinear Programming (MINLP) problem, if at least one of the functions f(x, y), g(x, y) or h(x, y) is nonlinear [44]

$$\min \left\{ f(\mathbf{x},\mathbf{y}) \;\middle|\; \begin{array}{lll} \mathbf{h}(\mathbf{x},\mathbf{y}) = 0 & \mathbf{h} : X \times U \to \mathbb{R}^{n_e} & \mathbf{x} \in X \subseteq \mathbb{R}^{n_c} \\ \mathbf{g}(\mathbf{x},\mathbf{y}) \geq 0, & \mathbf{g} : X \times U \to \mathbb{R}^{n_i}, & \mathbf{y} \in U \subseteq \mathbb{Z}^{n_d}. \end{array} \right\} \tag{1}$$

The vector inequality, g(x, y) $\geq$ 0, is to be read component-wise. Any vector $\mathrm{x}^{\mathrm{T}}\oplus$ satisfying the constraints of Equation (1) is called a feasible point of Equation (1). Any feasible point, whose objective function value is less or equal than that of all other feasible points is called an optimal solution. From this definition it follows that the problem might not have a unique optimal

solution.

The continuous variables in Equation (1) could for instance describe the states (temperature, pressure, etc.), flow rates or design parameters of plants or chemical reactors. The discrete variables, often binary variables, may be used to describe the topology of a process network or to represent the existence or non-existence of plants.

Table 1: Major types of optimization problems [44]

Type of problem	$f(\mathbf{x}, \mathbf{y})$	$\mathbf{h}(\mathbf{x}, \mathbf{y})$	$\mathbf{g}(\mathbf{x}, \mathbf{y})$	n_d
Linear Programming	$\mathbf{c}^{\mathrm{T}}\mathbf{x}$	$\mathrm{A}\mathbf{x} - \mathbf{b}$	$\mathbf{x}$	0
Mixed Integer LP	$\mathbf{c}^{\mathrm{T}}\mathbf{x}_{\oplus}$	$\mathrm{A}\mathbf{x}_{\oplus} - \mathbf{b}$	$\mathbf{x}_{\oplus}$	≥ 1
Mixed Integer NLP				≥ 1
Mixed Integer QP	$\mathbf{x}_{\oplus}^{\mathrm{T}}\mathrm{Q}_{\oplus} + \mathbf{c}^{\mathrm{T}}\mathbf{x}_{\oplus}$	$\mathrm{A}\mathbf{x}_{\oplus} - \mathbf{b}$	$\mathbf{x}_{\oplus}$	≥ 1
Nonlinear Programming				0
Global Optimization				≥ 0

Depending on the functions f(x, y), g(x, y), and h(x, y) in Equation (1) and matrix A of m rows and n columns, i.e., A $\in \mathcal{M}(m \times n, \mathbb{R}), \mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, and $n = n_c + n_d$ structured problem types can be defined as seen in Table 1. Real world problems lead to LP and MILP problems much more often than NLP or MINLP problems. QP deals with quadratic programming problems. They have a quadratic objective function but only linear constraints. QP and MIQP issues often arise in applications in the financial services industries. Since some problems arise as subproblems of others, it is very important that algorithms for solving subproblems are well understood and operated efficiently. While LP problems can be solved relatively easy (the number of iterations, and thus the effort to solve real-world LP problems with m constraints grows approximately linearly in m), the computational complexity of MILP and MINLP grows exponentially with n_d [44]. Numerical methods to solve NLP problems work iteratively and the computational problems are related to questions of convergence, getting stuck in bad local optima and availability of good initial solutions. Global optimization [65, 39, 51] applies to both NLP and MINLP problems and its complexity increases exponentially in the number of all variables entering

nonlinearly into the model [44].

Other notable cases of optimization problems are constraint satisfaction problems and multi-objective optimization. Strictly speaking, the constraint satisfaction problem is not an optimization problem. They lack an objective function. However, there are various goals associated with these issues. Find a solution, find all the solutions, find the outer and inner coverings for the solution, find N solutions that are different enough to be presented to the end user who chooses one of them, prove that no solution exists. All of these goals can be achieved partially or fully, even in the presence of rounding errors.

Multi-objective optimization [56], also known as multi-criteria optimization, handles problems involving multiple objective functions. A simple approach to solving a multi-criteria problem is to represent all goals on a common scale of goodness, but it's almost impossible. The problem is actually comparing different goals on a common scale. Basically, it can be distinguished between the two cases, either to find the Pareto optimal solution, or solve the problem individually for each objective function [44].

1.2 Examples of mathematical optimization problems

In order to better understand differences between different types of optimization problems few examples of the specific problems are provided.

First, an example of linear programming problem by Dantzig, G. B. [15] is given as seen in Equation (2). This classical transportation problem has the objective function to minimize the cost of transportation subject to demand and supply constraints. The transportation problem applies to situations where a single commodity is transported from various sources of supply (origins) to different demands (destinations).

Let there be m sources of supply $s_1, s_2, \ldots, s_m$ having a_i ($i = 1, 2, \ldots, m$) units of supplies, respectively, to be transported among n destinations $d_1, d_2, \ldots, d_n$ with b_j ($j = 1, 2, \ldots, n$) units of requirements, respectively.

Let c_{ij} be the cost for shipping one unit of the commodity from source i to destination j for each route. Suppose x_{ij} represents the units shipped per route from source i to destination j. In that case, the problem is determining the

transportation schedule that minimizes the total cost of satisfying the supply and demand conditions.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} \leq a_i, \quad \text{for } i = 1, 2, \ldots, m \text{ (supply constraints)}, \\
& \sum_{i=1}^{m} x_{ij} \geq b_j, \quad \text{for } j = 1, 2, \ldots, n \text{ (demand constraints)}, \\
& x_{ij} \geq 0, \quad \text{for all } i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n
\end{aligned}
\tag{2}
$$

MAGIC Power Scheduling Problem [16] seen in Equation (3) is an example of MIP problem where a number of power stations are committed to meet demand for a particular day. Different types of generators having different operating characteristics are available. Generating units can be shut down or operate between minimum and maximum output levels. Units can be started up or closed down in every demand block. Attempt is made to minimize the cost (amount of fuel) for power production.

$$
\begin{aligned}
\text{minimize} \quad & \text{cost} = \sum_{g,t}(\text{dur}_t \cdot \text{cmin}_g \cdot \text{n}_{g,t} + \text{st}_g \cdot \text{s}_{g,t} \\
& \quad + 1000 \cdot \text{dur}_t \cdot \text{cinc}_g \cdot (\text{x}_{g,t} - \text{minp}_g \cdot \text{n}_{g,t})) \\
\text{subject to} \quad & \sum_{g} \text{x}_{g,t} \geq \text{dem}_t, && \forall t, \\
& \sum_{g}(\text{maxp}_g \cdot \text{n}_{g,t}) \geq 1.15 \cdot \text{dem}_t, && \forall t, \\
& \text{s}_{g,t} \geq \text{n}_{g,t} - \text{n}_{g,t-1}, && \forall g, t, \\
& \text{x}_{g,t} \geq \text{minp}_g \cdot \text{n}_{g,t}, && \forall g, t, \\
& \text{x}_{g,t} \leq \text{maxp}_g \cdot \text{n}_{g,t}, && \forall g, t, \\
& \text{n}_{g,t} \in \mathbb{Z}_+, && \forall g, t, \\
& \text{s}_{g,t} \geq 0, && \forall g, t
\end{aligned}
\tag{3}
$$

In Equation (3) t represents demand blocks and g generators. Parameter dem_t defines demand in 1000mW blocks, dur_t defines duration of demand.

Next, parameters minp_g and maxp_g define minimal and maximal power of generator. Parameters cmin_g and cinc_g define minimal cost per hour and cost increase per mW. st_g defines starting fuel capacity and num_g defines number of units in generator. Variable $\text{x}_{g,t}$ calculates generator output, $\text{n}_{g,t}$ number of generators in use, and $\text{s}_{g,t}$ number of generators started up. Lastly, cost is a total operating cost which is attempted to minimize.

Fuel Scheduling and Unit Commitment Problem [78] seen in Equation (4) is a MINLP problem which addresses the problem of fuel supply to plants and determining on/off status of units simultaneously to minimize total operating cost. There are two generating units to meet a total load over a 6-hour period. One of the unit is oil-based and has to simultaneously meet the storage requirements, flow rates etc. There are limits on the generation levels for both the units.

$$
\begin{aligned}
\text{minimize} \quad & \text{cost} = \sum_t (300 + 6 \cdot \text{others}_t + 0.0025 \cdot (\text{others}_t)^2) \\
\text{subject to} \quad & \text{poil}_t \geq 100 \cdot \text{status}_t, && \forall t, \\
& \text{poil}_t \leq 500 \cdot \text{status}_t, && \forall t, \\
& \text{volume}_t = \text{volume}_{t-1} + 500 - \text{oil}_t + \text{initlev}_t, && \forall t, \\
& \text{oil}_t = 50 \cdot \text{status}_t + \text{poil}_t + 0.005 \cdot (\text{poil}_t)^2, && \forall t, \\
& \text{load}_t \leq \text{poil}_t + \text{others}_t, && \forall t, \\
& \text{status}_t \in \{0, 1\}, && \forall t, \\
& \text{status}_t \geq 0, && \forall t, \\
& \text{oil}_t \geq 0, && \forall t
\end{aligned}
\tag{4}
$$

In Equation (4) t represents scheduling periods. Parameters load_t and initlev_t define load level and initial level of the oil storage tank. Next, status_t represents on or off status of the oil based generating unit, poil_t is generation level of oil based unit, while others_t is generation level of other generation units. oil_t defines oil consumption and volume_t the volume of oil in the storage tank. Lastly, cost is a total operating cost which is attempted to minimize.

Social Accounting Matrix Balancing Problem [17] seen in Equation (5) is an example of QCP problem which captures all the circular flows in an

economy and is called balanced if the row totals equal the column totals. A sample problem illustrates attempt to balance such matrices.

$$\begin{aligned}
\text{minimize} \quad & \text{dev} = \sum_{i,j|\text{xw}_{i,j}} \left(\frac{\text{xw}_{i,j} \cdot ((\text{xb}_{i,j} - \text{x}_{i,j}))^2}{\text{xb}_{i,j}}\right) + \sum_{i|\text{tw}_i} \left(\frac{\text{tw}_i \cdot ((\text{tb}_i - \text{t}_i))^2}{\text{tb}_i}\right) \\
\text{subject to} \quad & \text{t}_i = \sum_{j|\text{xb}_{i,j}} \text{x}_{i,j}, \quad \forall i, \\
& \text{t}_j = \sum_{i|\text{xb}_{i,j}} \text{x}_{i,j}, \quad \forall j
\end{aligned} \tag{5}$$

In Equation (5) i and j represent accounts. Parameter $\text{xb}_{i,j}$ defines original estimates, $\text{xw}_{i,j}$ weights for cells, tb_i original totals, and tw_i weights for totals. Next, variable $\text{x}_{i,j}$ represents estimated cells and t_i estimated totals. Lastly, dev is a deviation between the two accounts which is attempted to minimize.

In the following sections, research is focused on algebraic modeling languages, the way they enable modeling optimization problems, and help solving them using specific solver algorithms.

1.3 Algebraic modeling languages

Algebraic modeling languages (AMLs) are sophisticated software packages that provide a key link between an analyst's mathematical conception of an optimization model and the complex algorithmic routines that seek optimal solutions. By allowing models to be described in the high-level, symbolic way that people think of them while automating the translation to and from the different low-level forms required by the algorithms, algebraic modeling languages significantly reduce the effort and increase the reliability of formulation and analysis [25].

The first algebraic modeling languages, developed in the late 1970s, were game-changers. They allowed separating the model formulation from the implementation details [44] while keeping the notation close to the problem's mathematical formulation [27]. Since the data appears to be more volatile than the problem structure, most modeling language designers insist on the data and model structure being separated [41]. Therefore, the central idea in

modern AMLs is the differentiation between abstract models and concrete problem instances [37]. A specific model instance is generated from an abstract model using data. This way, the model and data together specify a particular instance of an optimization problem for which a solution can be sought. This is realized by replicating every entity of an abstract model over the different elements of the data set. Such a feature often is referred to as a set-indexing capability of the AML [27].

To provide a better understanding of what aspects of mathematical modeling are being addressed by the algebraic modeling language, it was chosen to use an example of the classical linear programming transportation problem provided in Equation (2).

First, a concrete instance of the problem is taken and it is shown what knowledge has to be captured to create an algebraic model of such an instance. Later, it is shown how this would be modeled in two widely used algebraic modeling languages `AMPL` and `JuMP`.

1.3.1 An instance of a concrete problem

A general problem statement was defined in the Equation (2), however, to understand what has to be modeled, it is needed to look into a concrete instance of the problem. In this case, an example where the goal is to minimize the cost of shipping goods from two plants to three markets, subject to supply and demand constraints is used. Listing 1 provides the characteristics of such a model instance in a mathematical format.

First, it is needed to define the sets and indices describing plants and markets. Then, the parameters that will hold the data about the capacity of each plant, the demand at each market, and the distance between them have to be defined. Later, is has to be described how the transportation cost per case should be calculated. Having the data initiated, two variables excepted to be calculated by the solver have to be defined: the shipment quantity between the plants and markets, and the total transportation cost. Lastly, it is needed to describe the demand and supply constraints and the objective function.

Sets:	Plants	{Seattle, San Diego} ($m = 2$ plants)
	Markets	{New-York, Chicago, Topeka} ($n = 3$ markets)
Indices:	i	On plants
	j	On markets
Parameters:	A_i	Capacity of plant i
	B_j	Demand at market j
	F	Freight in dollars per case per thousand miles
	D_{ij}	Distance in thousands of miles
	C_{ij}	Transport cost in thousands per cases ($F * D_{ij}/1000$)
Variables:	X_{ij}	Shipment quantities in cases
	Z	Total transportation costs in thousands of dollars
Constraints:	$\sum_{j=1}^{n} X_{ij} \leq A_i$; for $i = 1, 2, \ldots, m$; observe supply limit at plant i	
	$\sum_{i=1}^{m} X_{ij} \geq B_j$; for $j = 1, 2, \ldots, n$; satisfy demand at market j	
	$X_{ij} \geq 0$; for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$	
Objective:	Minimize transportation cost subject to supply and demand constraints	
	Min Z $= \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij}$	

Listing 1: Concrete model instance of the classical transportation problem by Dantzig, G. B. [15] with the goal of minimizing the shipping cost of goods from two plants to three markets.

1.3.2 Formulation of a concrete problem using AML

Once it is known how the concrete model instance of an optimization problem should look like (see Listing 1), it is needed to transfer such mathematical concepts to the concepts of a specific algebraic modeling language. Listing 2 demonstrates how such an instance of a problem is expressed in `AMPL` algebraic modeling language. If it would be needed to define the model in a different algebraic modeling language than `AMPL`, it would require to start from scratch and take the mathematical concepts to yet another textual notation of another algebraic modeling language. An example of how the same model would be described in `JuMP` format can be seen in Listing 3.

```
set I;
set J;
param a{i in I};
param b{j in J};
param d{i in I, j in J};
param f;
param c{i in I, j in J} := f * d[i,j] / 1000;
var x{i in I, j in J} >= 0;
minimize cost: sum{i in I, j in J} c[i,j] * x[i,j];
s.t. supply{i in I}: sum{j in J} x[i,j] <= a[i];
s.t. demand{j in J}: sum{i in I} x[i,j] >= b[j];

data;
set I := Seattle San-Diego;
set J := NNew-YorkChicago Topeka;
param a := Seattle     350
San-Diego   600;
param b := New-York    325
Chicago     300
Topeka      275;
param d :              New-York   Chicago   Topeka :=
Seattle     2.5        1.7        1.8
San-Diego   2.5        1.8        1.4  ;
param f := 90;
end;
```

Listing 2: The Classical Transportation Problem by Dantzig, G. B. defined in AMPL format.

Even not going to the deeper analysis, it can be observed that the syntax of algebraic modeling languages differs. In the two given examples AMPL seems to be the one targeted towards practitioners with a mathematical background, while JuMP is more catering for other practitioners who are more familiar with common programming languages. The differences between AMLs will be explored in more detail in the comparative analysis section. Examples of how this problem is modeled using GAMS and Pyomo can be found in Appendix A.

```
using JuMP
ORIG = ["Seattle", "San-Diego"]
DEST = ["New-York", "Chicago", "Topeka"]
supply = [350, 600]
demand = [325, 300, 275]
cost = [
        2.5   1.7   1.8;
        2.5   1.8   1.4
]
F = 90
cost_f = [F * cost[i,j] / 1000
        for i in 1:length(ORIG), j in 1:length(DEST)]

model = Model()

@variable(model, trans[1:length(ORIG), 1:length(DEST)] >= 0)
@objective(model, Min, sum(cost_f[i, j] * trans[i, j]
        for i in 1:length(ORIG), j in 1:length(DEST)))
@constraint(model, [i in 1:length(ORIG)],
        sum(trans[i, j] for j in 1:length(DEST)) <= supply[i])
@constraint(model, [j in 1:length(DEST)],
        sum(trans[i, j] for i in 1:length(ORIG)) >= demand[j])
```

Listing 3: The Classical Transportation Problem by Dantzig, G. B. defined in JuMP format.

1.4 Essential characteristics of AMLs

In order to identify the most prominent algebraic modeling languages for mathematical optimization, and evaluate how modern AMLs match the current needs of practitioners, it is needed to understand what essential features and characteristics they should provide. Kallrath, J. defines the essential characteristics of a modern AML in the following way [44]:

1. Problems are represented in a declarative way, i.e. specifying the problem's properties: space, set of constraints, and optimality requirements.

2. There is a clear separation between problem definition and the solution process.

3. There is a clear separation between the problem structure and its data.

Besides that, the support for mathematical expressions and operations needed

for describing nonlinear models is often considered an important feature of an AML [44]. Moreover, it is worth observing that most interpreters included in today's AMLs are based on automatic differentiation [27], a process in which the modeling language can compute the derivatives of problems from the model description without the assistance of the user [44]. This motivates to include automatic differentiation as an additional, important feature of a modern AML.

The algebraic expressions are useful in describing individual models and describing the manipulation of models and transformation of data. Thus, almost as soon as AML became available, users started finding ways to adapt model notations to implement sophisticated solution strategies and iterative schemes. These efforts stimulated the evolution within AMLs of scripting features, including statements for looping, testing, and assignment [25]. Therefore, scripting capabilities are an integral part of AMLs.

1.5 Most prominent AMLs

Based on the identified essential characteristics of modern AMLs and criteria defined below four most prominent AMLs have been chosen to be analyzed futher: `AMPL`, `GAMS`, `JuMP`, and `Pyomo`. The selection of AMLs was based on the following criteria:

- AMLs which won the 2012 INFORMS Impact Prize award[2] dedicated to the originators of the five most important algebraic modeling languages: `AIMMS`, `AMPL`, `GAMS`, `LINDO/LINGO`, and `MPL`;
- the popularity of AMLs based on NEOS Server model input statistics for the year 2020[3];
- open-source options that are attractive for the academic society or in situations where budgets are tight.

It was chosen to include `GAMS` and `AMPL` based on NEOS Server popularity, respectively, with a 75% and 18% share of jobs executed via the

[2] https://www.informs.org/About-INFORMS/News-Room/Press-Releases/INFORMS-Impact-Prize-2012

[3] NEOS Server. Solver Statistics: https://neos-server.org/neos/report.html

NEOS platform between January and September of 2021. More details in Table 2.

Table 2: Number of jobs by AML type submitted to NEOS platform between January and September of 2021

AML	**# jobs**	**% share**
AMPL	732756	75.49%
GAMS	180740	18.62%
LP	15857	1.63%
TSP	11012	1.13%
MPS	10265	1.06%
...	...	...
Total	970657	

`JuMP` and `Pyomo` were included as the most prominent open-source AMLs. `AIMMS` was excluded since all of attempts to acquire a trial or academic license were unsuccessful. It was decided to exclude `MPL` since it has not been updated for the last five years. `LINGO` was excluded as a solver-specific modeling language.

1.6 Related literature review

There has not been any recent bibliometric analysis of state of the art in algebraic modeling languages. Therefore, it was decided that analyzing the current body of knowledge in algebraic modeling language research through a bibliometric study is needed. Bibliometric studies can help us understand the extent of a topic, emergent trends, and its evolution through time.

This section aims to analyze the metadata of papers and articles published between the year 2000 and the year 2021 related to the four algebraic modeling languages chosen for the research.

1.6.1 Methodology

In order to identify the key publications on algebraic modeling languages, the adopted version of PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) methodology [60] for a literature search in scientific databases was followed. For the analysis, it was chosen to use two well-known databases, *Web of Science* (WoS) administered by Clarivate Analytics and *Scopus* administered by Elsevier. The search was performed in titles, abstracts, author keywords, and keywords extracted by indexing services (e.g., KeyWords Plus in WoS). Only papers published in conferences, workshops, books, and journals related to the research topic were considered.

To find relevant publications for the research, the following search string was used:

```
((GAMS OR AMPL OR Pyomo OR (JuMP AND Julia)) AND
("algebraic model*ing" OR "optimi*ation"))
```

The search string was designed to find all publications where either of the four chosen AMLs are mentioned, while references to algebraic modeling or optimization are also made. It was required to fine-tune the search string to accommodate for the differences between American and British English in the spelling of words modeling, and optimization. It was also required to narrow down the occurrence of the word JuMP to always include the word Julia (the programming language JuMP uses) in the match since using the general word *jump* resulted in tens of thousands of irrelevant matches.

Searches in Web of Science produced 1568 potentially relevant publications, while Scopus resulted in 2125 potentially relevant records. However, both databases included the same retracted paper and had duplicates in the results. That was especially visible in Scopus, having 53 duplicates out of which 45 were the same papers published in the same sources and for some reason indexed multiple times. In 8 instances, Scopus had duplicates of the same papers published in different sources.

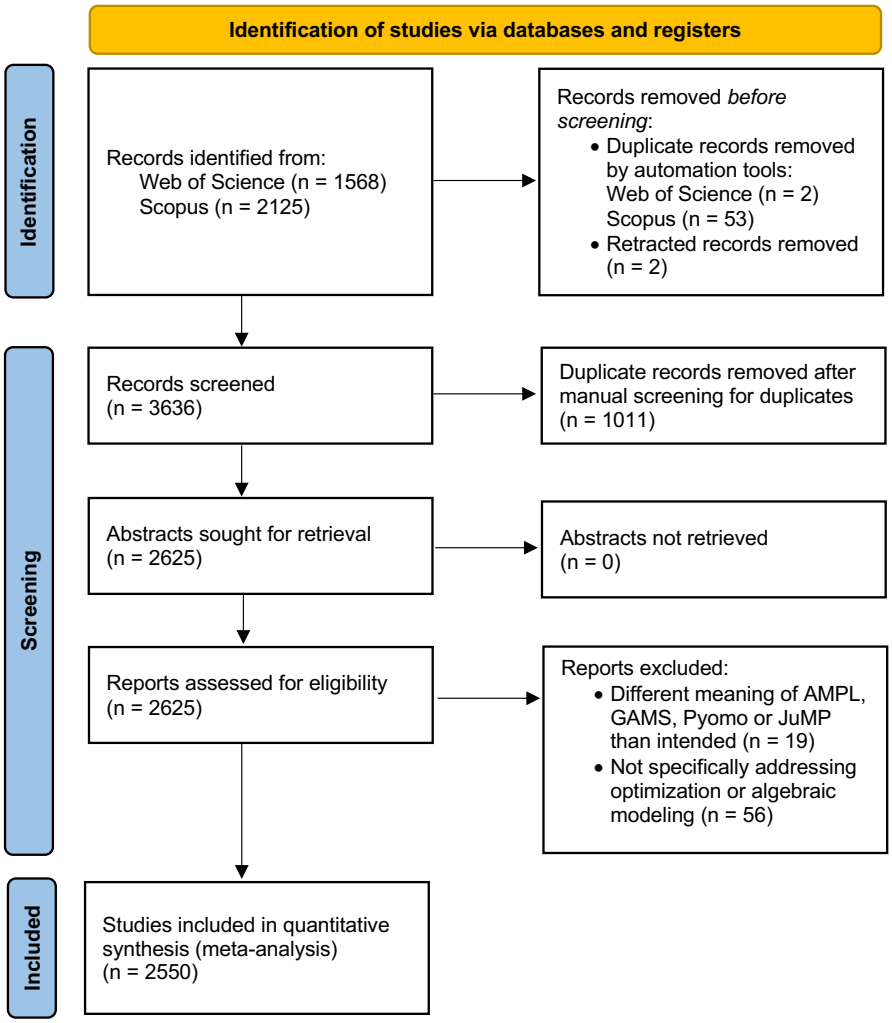

Figure 1: PRISMA 2020 flow diagram [60] for bibliometric analysis of scientific literature about `AMPL`, `GAMS`, `JuMP` and `Pyomo` algebraic modeling languages.

After merging the Web of Science and Scopus data sets, a total of 3636 records were manually screened for potential duplicates, finding 1011 and leaving us with 2625 relevant sources. The manual screening was required since automation tools produced too many false positives once merging two data sets. Later, the abstracts of records were analyzed and assessed for eligibility based on relevance in the field of algebraic modeling and specific

application of AMPL, GAMS, JuMP, and Pyomo in the research. This part of the screening removed 19 reports where terms AMPL, GAMS, JuMP, and Pyomo were used in a different context than expected, and 56 reports were optimization or algebraic modeling was not a primary focus area.

Once the screening process was completed, a data set of 2550 documents available for quantitative synthesis to systematically review meta-data using statistical methods was present. A PRISMA 2020 Flow Diagram of the literature search and selection process is presented in Figure 1.

After retrieving and selecting the papers, there was a need to clean up and unify the meta-data originating from different citation databases. WoS and Scopus databases use a very different approach to codify the bibliographic metadata. Thus, it was chosen to use a style-independent text-based file format, BibTeX, for merging the lists of bibliographies. A reference management software Zotero was chosen to help in this task, but also it was required to create some shell scripts using regular expressions to manipulate the content of the list. Mostly, it was needed to merge indexed keywords from WoS, called KeyWords Plus, with indexed keywords available in Scopus and unify how the number of times a paper has been cited was encoded. Moreover, it was required to unify the way correspondence address is presented to be able to extract country information. Attempt of unifying the way citations present in the paper are encoded wasn't successful. WoS applies a preprocessing analysis on reference lists, rewriting references as first author, year, journal, issue, DOI, while Scopus stores the full APA record as the author included it in his manuscript. Lack of unification means that, at the moment, it is not possible to do citation and co-citation analyses on the merged databases.

To help in the statistical analysis of the data set, it was chosen to use R (programming language) package created by Massimo Aria and Corrado Cuccurullo called Bibliometrix [4]. It is an open-source tool for quantitative research in scientometrics and bibliometrics that includes all main bibliometric methods of analysis. This package supports bibliographic database files from Web of Science, Scopus, Dimensions, The Lens, PubMed, and Cochrane Library. It also allows performing bibliometric analysis and building data matrices for co-citation, coupling, scientific

collaboration analysis, and co-word analysis.

In addition to bibliometric analysis, a graphical analysis of the bibliographic material using VOSviewer software [76] was added. This software collects data and generates maps based on bibliographic coupling, co-authorship, citation, co-citation, and co-occurrence of keywords. Density visualization can provide a quick overview of the main areas in a bibliometric network. Overlay visualization can be used to show development over time.

1.6.2 Findings of literature review

A total of 2550 documents, published in 927 sources (journals, books, etc.) was analyzed. These documents were (co-)authored by 4761 people. The vast majority of the documents are multi-authored, only 133 documents being single-authored. The average number of authors per document is 1.87.

Table 3 describes the distribution of publications based on document types. It can be observed that the majority of publications (56%) are articles published in journals while proceeding and conference papers follow next. Books and book chapters account for only 4.5% of publications.

Table 3: Distribution of publications by document type.

#	Document type	# documents
1	Article	1441
2	Proceedings paper	500
3	Conference paper	494
4	Book chapter	108
5	Book	7

According to Figure 2 the number of publications related to the four chosen AMLs and algebraic modeling has been growing continuously from the year 2000 to the year 2019, having an annual percentage growth rate of 12.3%. However, in recent years, 2020 and the first 9 months of 2021, a change in publication trends, where numbers are decreasing (from 284 publications in 2019 to 234 in 2020) can be seen. The decreasing growth rate could indicate that research in this field is consolidating. However, to

confirm the hypothesis, it is needed to analyze in more detail which main research areas the publications are covering.

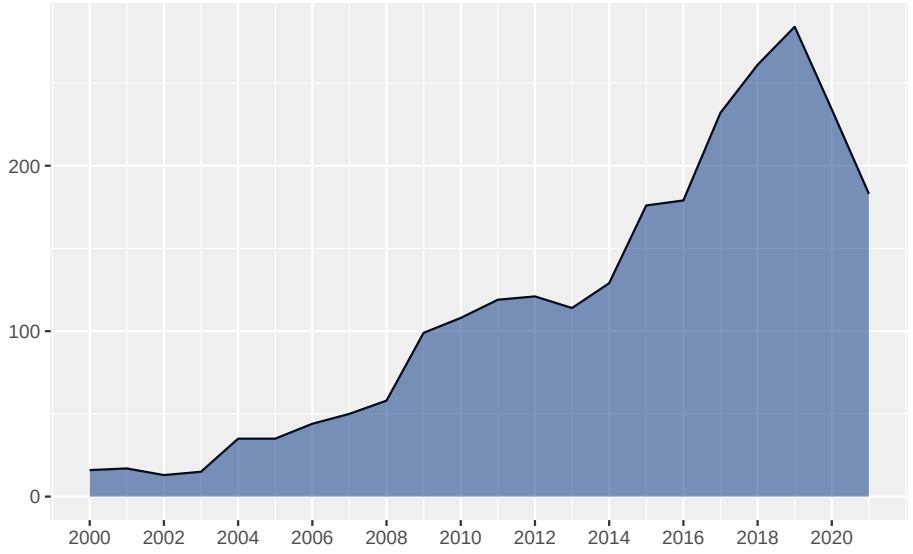

Figure 2: Annual scientific production as publications per year in the field of algebraic modeling languages.

Since Web of Science and Scopus data sources were merged, it was not possible to use the research areas WoS assigned to each paper, since its only available in the WoS database. However, an alternative approach was chosen to use data about the sources (journals, books, etc.) where papers have been published, thus allowing to identify a research area based on the scope of the source.

Table 4 provides an overview of the most relevant publication sources indicating how many of the publications were published in particular sources. Out of the top ten sources, 8 are journals and 2 are book series. Elsevier is a publishing company for 6 out of the top 10 sources. The majority of journals and book series are concentrated around two main research areas: chemical engineering and energy engineering. Thus it can be concluded that in most of the cases `AMPL`, `GAMS`, `JuMP`, and `Pyomo` algebraic modeling languages are used as the basis for computer science applications for solving chemical or energy engineering problems. Another emerging field of research is focusing

on renewable energy, sustainability, and the environment. This can confirm the hypothesis that the decrease of publications within the last 2 years can be the result of research in chemical and energy engineering consolidating while a shift towards renewable energy research still being on the verge.

Table 4: Top ten most relevant publication sources. The number of articles in each source.

#	Sources	# articles
1	Computer Aided Chemical Engineering	78
2	Computers & Chemical Engineering	42
3	Energy	40
4	Industrial & Engineering Chemistry Research	35
5	Journal of Cleaner Production	30
6	Chemical Engineering Transactions	29
7	International Journal of Electrical Power & Energy Systems	29
8	Energies	24
9	Springer Optimization And It's Applications	19
10	Applied Energy	18

Figure 3 shows that Iran is the country whose authors have published most documents, followed by China and USA. The ten first countries accumulate 58% of the articles published related to the chosen field of research. Iran, China, and the USA are the top contributors correlating well with the nation-state strategies set to ensure independent and efficient production of energy, thus requiring to develop and apply optimization models in the fields of chemical and energy engineering.

The international collaboration among the top 10 countries is the lowest in India and China, with a multiple country publications rate of 8% and 10%, respectively. The international collaboration reaches a maximum in the United Kingdom, where 40% of the papers are of this kind.

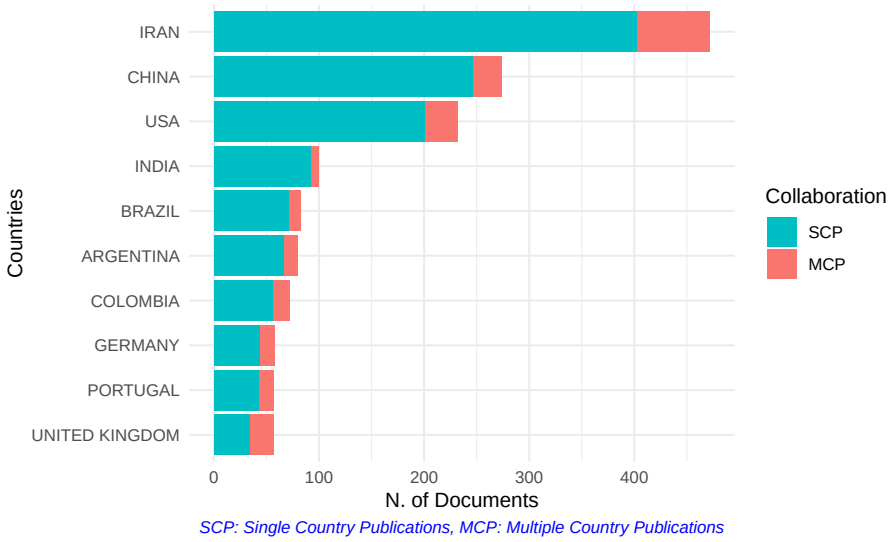

Figure 3: Top ten countries by annual scientific production. Single and multiple country publications split.

Table 5: Top ten countries by citations of scientific production. Total citations and average citations per article.

#	Country	Total citations	Average citations
1	Iran	4745	10.053
2	USA	3283	14.151
3	China	1695	6.186
4	Canada	868	15.500
5	Turkey	855	15.268
6	Argentina	783	9.787
7	Germany	771	13.293
8	Spain	744	14.880
9	Portugal	734	12.877
10	Brazil	594	7.157

Table 5 displays the main countries, ordered by total citations. The overall average number of citations per article is 8.5. Iran and USA, the two countries with the most published articles, also have the most total citations

and are above the average in the number of citations figures, with 10.053 and 14.151, respectively. Even though China is the second country in terms of published articles and the third in total citations, it still has the lowest average citations per article among the leading countries in total citations (6.186). This can indicate the difference in scientific importance or quality of the articles.

Table 6 lists the ten most cited papers indexed by WoS and Scopus. The table also shows what is the average number of citations per year and identifies which of the AML keywords it contains.

Paper titled "JuMP: A Modeling Language for Mathematical Optimization" written by Dunning et al. [18] has been identified as the most cited paper with 400 total and an average of 80 citations per year. The popularity of the paper validates the decision to include `JuMP` as one of four AMLs in the scope of the research.

The most cited paper based on average citations per year is "CasADi: a software framework for nonlinear optimization and optimal control" published by Andersson et al. [3] having 103 yearly citations. The paper proposes a self-contained symbolic framework that can be used to model and solve optimization problems constrained by differential equations, i.e., optimal control problems. In essence, it allows the user to construct symbolic expressions using a `MATLAB` inspired everything-is-a-matrix syntax as an alternative to the approaches adopted by AMLs. The popularity of such a paper also motivates the need to explore alternative options in modeling language landscape.

Seven out of the ten most cited papers were mentioning `GAMS` as one of the keywords. However, in the majority of the cases `GAMS` was used as the tool to solve an optimization problem, while the paper focused on a general problem in the chemical or energy engineering field. `AMPL` and `JuMP` were mentioned as keywords twice, while `Pyomo` was not mentioned at all amongst the top ten cited papers. Thus it can be concluded `GAMS` being the most adopted AML among the top-cited papers in the field of research.

Table 6: Top ten most cited papers. Ordered by total citations.

Source	Title	Keywords	Total Citations	Citations per Year
Dunning et al. (2017)	JuMP: A Modeling Language for Mathematical Optimization	JuMP	400	80.00
Yetilmezsoy et al. (2009)	Response surface modeling of Pb(II) removal from aqueous solution by Pistacia vera L.: Box–Behnken experimental design	AMPL	397	30.54
Ugray et al. (2007)	Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization	GAMS	385	25.67
Morais et al. (2010)	Optimal scheduling of a renewable micro-grid in an isolated load area using mixed-integer linear programming	GAMS	334	27.83
Andersson et al. (2019)	CasADi: a software framework for nonlinear optimization and optimal control	AMPL,GAMS, JuMP,Pyomo	309	103.00
Kannan et al. (2010)	A genetic algorithm approach for solving a closed loop supply chain model: A case of battery recycling	GAMS	229	19.08
Saeedi et al. (2019)	Robust optimization based optimal chiller loading under cooling demand uncertainty	GAMS	193	64.33
Zeineldin et al. (2006)	Optimal coordination of overcurrent relays using a modified particle swarm optimization	GAMS	185	11.56
Khodaei et al. (2018)	Fuzzy-based heat and power hub models for cost-emission operation of an industrial consumer using compromise programming	GAMS	184	46.00
Ferris and Munson (2000)	Complementarity Problems in GAMS and the Path Solver	GAMS	184	8.36

Table 7 shows the ten most used author keywords in the data set of the bibliometric analysis. Web of Science and Scopus provide two types of keywords: author keywords, which are those provided by the original authors, and extracted keywords like Keywords-Plus (WoS) or Indexed Keywords (Scopus), which are those generated from the titles and abstracts of the papers by the citation database algorithms. Table 8 shows the top ten extracted keywords (Keywords Plus and Indexed Keywords) in the data set.

Table 7: Top ten most used author keywords. The number of articles containing a keyword is provided.

#	**Author keywords**	**# articles**
1	Optimization	505
2	GAMS	247
3	Mathematical Programming	91
4	MINLP	83
5	Uncertainty	82
6	Microgrid	73
7	Multi-objective Optimization	72
8	Demand Response	69
9	Nonlinear Programming	66
10	Stochastic Programming	60

The most frequent keyword in both cases is *Optimization*, which is not surprising. However, if looking into the following keywords, remarkably, keyword *GAMS* is the second most frequent in author keywords, but it does not appear as the extracted keyword at all. First of all, it shows the importance and popularity of `GAMS` which makes it stand out amongst other AMLs. Secondly, it points towards GAMS being an important tool used by scientists to solve optimization problems but shows limitations of the algorithms used to extract keywords, where they do not discriminate such a detailed keyword. On the contrary, keyword extraction algorithms are precise at identifying general keywords such as *Model*, *Design*, or *System*, as many engineering papers are focused on these aspects of the application of AMLs.

It is also relevant to notice that *Microgrid* is a popular keyword used by the authors, once more arguing for the importance of AMLs in the energy engineering field of research.

Table 8: Top ten most used extracted keywords: Keywords Plus in WoS and Indexed Keywords in Scopus. The number of articles containing a keyword is provided.

#	**Extracted keywords**	**# articles**
1	Optimization	614
2	Integer Programming	197
3	Model	158
4	Design	154
5	Management	132
6	Algorithm	126
7	System	116
8	Nonlinear Programming	102
9	Systems	100
10	Costs	90

Figures 4 and 5 were generated using VOSviewer software [76], which allows performing keyword co-occurrence analysis, i.e., counting the number of documents in which particular keywords appear together and visualizing the links between them in cloud maps.

It was chosen to use fractional counting for the strength of co-occurrence where the weight of the links between keywords is fractionalized. The idea of fractional counting is to reduce the influence of documents with many keywords. When fractional counting is used, the strength of a co-occurrence link between two keywords is determined not only by the number of documents they appear in together but also by the total number of keywords in each of the co-occurring documents.

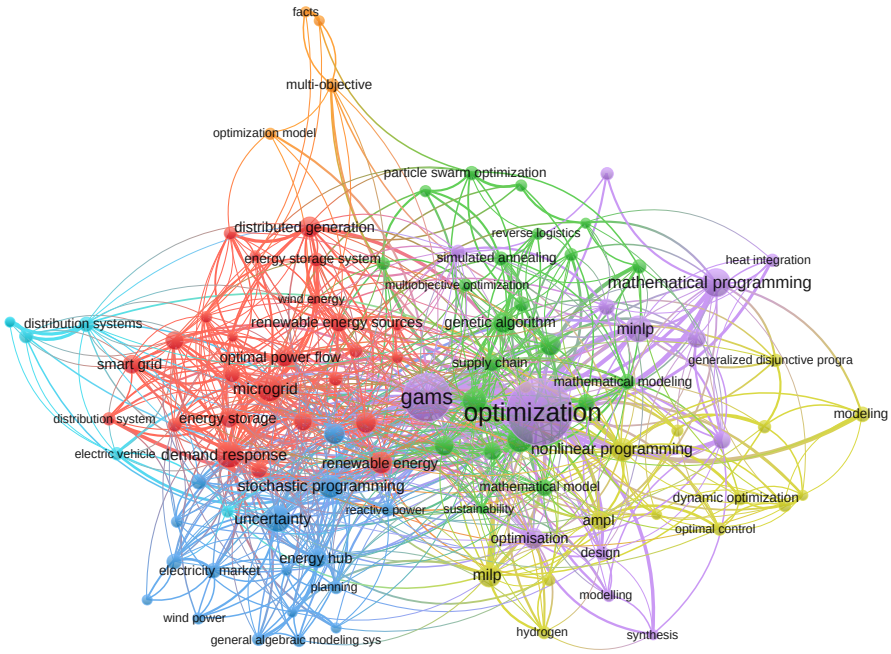

Figure 4: Cloud map of author keywords co-occurrence using fractional counting. Keywords with a minimum of 10 occurrences included.

There was a total of 5721 keywords appearing in 2550 papers in the bibliometric data set. Using VOSviewer, the minimum required number of occurrences for the keyword to be included in the cloud map was set to be 10. This resulted in 102 keywords being organized into clusters and visualized.

Figure 4 represents the cloud map with co-occurrence of keywords, showing how many times the keywords appear in the articles and how related they are to each other. The main finding is that the cloud could be divided into two parts. The right side is more related to optimization, mathematical programming, and modeling (purple, green, and yellow) and the left side is more related to energy and power engineering (red and blue). This allows to state that two primary types of papers exist: one focusing on solving energy engineering problems using algebraic modeling languages, and the other one investigating the algebraic modeling languages in the field of general mathematical modeling and optimization. A strong link between *optimization* and *GAMS* keywords appearing together was also observed, confirming previous statements of GAMS being the most prominent AML in the scientific

production analyzed.

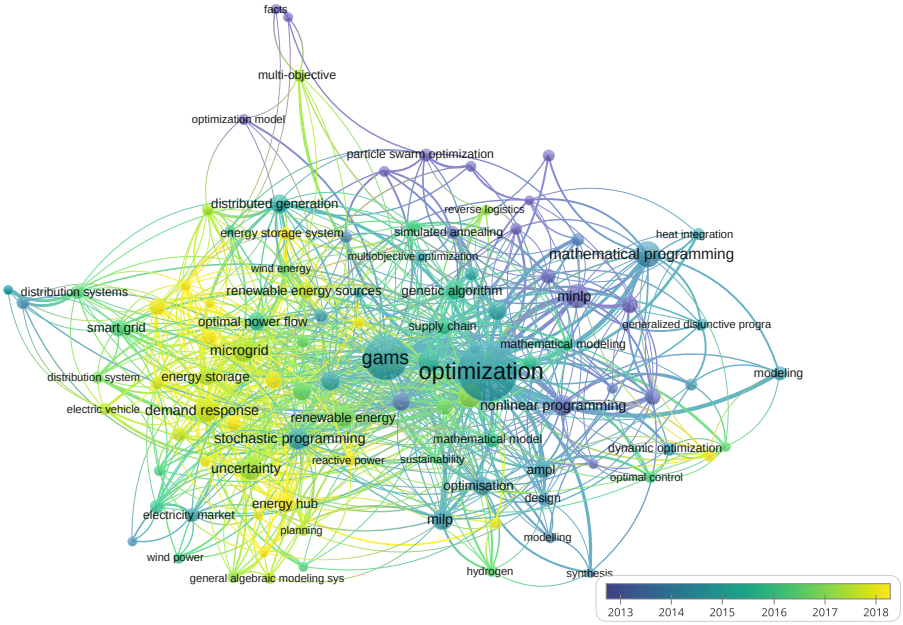

Figure 5: Cloud map of author keywords co-occurrence using fractional counting and yearly overlay. Keywords with a minimum of 10 occurrences included.

Figure 5 represents the cloud map for the keywords co-occurrence in the same data set, additionally using overlay visualization, which shows the development of keyword co-occurrence over time. Some of the emerging trends can be identified from this. First of all, clusters on the left-hand side indicating the number of publications focusing on renewable energy are growing over the time. This fits well with increasing interest in sustainability all over the world. On the right-hand side, a cluster of publications related to optimization, mathematical programming, `GAMS`, `AMPL` where the most contributions were done between 2013 and 2016 can be seen. This correlates well with the earlier statement of research in this field is consolidating. Thus, it is possible to state that the engineering of sustainable energy will be the driving force behind the near-future developments in the research field of mathematical optimization and algebraic modeling languages.

1.7 Conclusions

This chapter described mathematical optimization and algebraic modeling languages and identified the main characteristics of AMLs that will be considered in further chapters.

First, concept of mathematical optimization was introduce and types of optimization problems were discussed. Following, the significance of algebraic modeling languages was highlighted. Next, a concrete instance of the classical transportation problem by Dantzig [15] was described in logical concepts and how they are translated to the notations of two algebraic modeling languages `AMPL` and `JuMP`. Afterward, the key characteristics of modern AMLs were outlined, and the most prominent algebraic languages were identified and chosen for further theoretical and experimental research. Finally, to understand the extent of a topic, the emergent trends, and its evolution through time, a bibliometric analysis on the chosen AMLs was conducted. Papers and articles published between the year 2000 and the year 2021 were analyzed using the adopted version of PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) methodology [60] for a literature search in scientific databases. A total of 2550 documents, published in 927 sources (journals, books, etc.) was analyzed.

The following conclusions have been made:

1. AMLs allow separating the model formulation from the implementation details while keeping the notation close to the problem's mathematical formulation. It differentiates between abstract models and concrete problem instances, where a specific model instance is generated from an abstract model using data. This allows constructing reusable models and makes mathematical optimization more acceptable for practitioners without a deep mathematical background.

2. It can observed that the syntax of algebraic modeling languages differs. In the two given examples `AMPL` seems to be the one targeted towards practitioners with a mathematical background, while `JuMP` is more

catering for other practitioners who are more familiar with common programming languages.

3. Three key characteristics of AMLs are having problems represented in a declarative way, having a clear separation between the problem definition and the solution process, and having a clear separation between the problem structure and its data. The need for other features such as automatic differentiation or scripting is also evident, but not essential. Based on the identified requirements, academic recognition, and usage popularity, the four most prominent AMLs have been identified: `AMPL`, `GAMS`, `JuMP`, and `Pyomo`.

4. A constantly growing publication rate in the research field between the years 2000 and 2019 has been observed. However, in recent years, the numbers are decreasing, which together with the author keywords co-occurrence analysis indicates that research in this field is consolidating. Seven out of the ten most cited papers were mentioning `GAMS` as one of the keywords. `GAMS` was also between the top overall keywords which shows the importance and popularity of `GAMS`. In general, judging by the trends of the keywords, it can be predicted that the engineering of sustainable energy will be the driving force behind the near-future developments in the research field of mathematical optimization and algebraic modeling languages.

2 COMPARATIVE ANALYSIS OF ALGEBRAIC MODELING LANGUAGES

In the following section, investigation on how each of the chosen languages meets the requirements for a modern AML defined in the previous section is conducted. The websites of the AMLs and vendor documentation were used for this comparison. Any support of the identified features and capabilities were validated against the documentation the suppliers of the AMLs provide. Besides, an in-depth survey concluded by Robert Fourer in Linear Programming Software Survey [26] was also used as a reference. Later on, a more practical comparison of AML characteristics is conducted to identify the potential ease of use of AML in daily work.

2.1 Overview of existing AML software

Existing AML software for optimization modeling systems consist of the algebraic modeling language (AML) and environment (tools/interfaces) supporting it.

Modeling environments mediate between human modelers and solvers, providing general and intuitive ways to express symbolic models while offering features for importing data, generating problem instances, invoking solvers, analyzing results, scripting extended algorithmic schemes, and interfacing to broader applications. Software of this latter kind are typically built around a computer modeling language, either designed specifically for describing optimization models or adapted from the features of an already popular programming language. Solver software takes an instance of a model as input, applies a combination of algorithmic methods designed to find solutions that are optimal (or reasonably close to optimal), and returns the results.

Commercial and practical aspects of using the most popular modeling environments were assessed based on the following criteria:

- supported modeling languages;
- inter-language compatibility capabilities;

- support for parallelism;
- licensing and pricing;
- supported operating systems;
- UI technology.

A theoretical comparison of the before-mentioned aspects across four AMLs providing modeling environments: `AMPL`, `GAMS`, `JuMP` and `Pyomo` is described next.

In Table 9 the general features as provided in the website of the vendors are reviewed. It can be observed that all except one AML provide GUI for writing a textual model code and running some of the standard commands. `Pyomo` does not have any graphical interface at all.

It is also useful to observe that all of them support all three major operating systems, so the usage should be relatively smooth independent of what operating system or hardware modeler is using.

Table 9: Comparison of the basic features provided by AML software.

Software	**Vendor**	**Released**	**Operating System**	**UI**
AMPL	AMPL Optimization Inc.	1985	Windows Linux MacOS	GUI (Textual)
GAMS	GAMS Development Corp.	1978	Windows Linux MacOS	GUI (Textual)
JuMP	NumFOCUS	2017	Windows Linux MacOS	GUI (Textual)
Pyomo	COIN-OR	2008	Windows Linux MacOS	Command line

In Table 10 a review of which algebraic modeling languages each environment supports is provided. Here only languages that are supported without any additional plugins or extensions to the environment were

included. There are special tools like `GAMS Convert` that allow converting from one language to another, but only in scalar format. It was also found that a tool like `GAMS Convert` is the only way to support inter-language compatibility in most of the modeling environments.

Table 10: Comparison of supported algebraic modeling languages. Check mark (✓) states that AML software in the given column supports AML language in the given line.

	AMPL	**GAMS**	**JuMP**	**Pyomo**
.mod (AMPL)	✓			✓
.gms (GAMS)		✓		✓
.jl (Julia)			✓	
.py (Pyomo)				✓

Table 11: Pricing of the reviewed AML software. Commercial, trial, and academic licensing compared.

	AMPL	**GAMS**	**JuMP**	**Pyomo**
Commercial	Base: 4.000 USD Solvers: 2.000 - 10.000 USD	Base: 3.200 USD Solvers: 1.600 - 12.800 USD	Free	Free
Trial	Free for 30 days with unlimited features	Free, but restricted on model size and free solvers only	Free	Free
Academic	Base: 400 USD Solvers: 200 - 600 USD	Base: 640 USD Solvers: 320 - 1.920 USD	Free	Free

In Table 11 the pricing of the reviewed AML software is compared split into commercial, academic, and trial use scenarios. It can be seen that for commercial software even the bare minimum (base) version is costly while adding the cost of solvers can make the price rise significantly higher. Trial versions are quite limited for all of them and even the academic licensing does

not come for free.

2.2 Comparative analysis of the features

First, analysis on how the selected AMLs satisfy the three essential characteristics defined in [44] was done. The following AML characteristics were reviewed:

- Are the problems represented in a declarative way, by specifying the problem's properties: space, set of constraints, and optimality requirements?
- Does a clear separation between problem definition and the solution process exist?
- Does a clear separation between the problem structure and its data exist?

In all reviewed AMLs, the optimization problems are represented in a declarative way. Furthermore, since all of them are part of a specific modeling system, a clear separation between problem definition and the solution process in the context of the modeling system exists. The separation between the problem structure and its data is supported in all reviewed languages. It should be noted that `GAMS`, `JuMP`, and `Pyomo` also allow initiating data structures during their declaration, while `AMPL` only support it as a separate step in the model instance building process. However, while it might be convenient for building a simple model, the lack of direct data structure initiation should not be considered as an advantage, since in real-world cases it is rarely needed. Therefore, it can be concluded that all reviewed languages fulfill the essential characteristics of modern AMLs.

2.2.1 Support for general features

Table 12, provides an overview of the key features each AML supports. For creating such a summary, the information provided by the AML vendors on their websites was used. All reviewed AMLs allow modeling problems in a solver-independent manner. Additionally, `JuMP`, and `Pyomo` provide a more powerful way to define advanced algorithms using Julia, or Python

Table 12: An in-depth overview of features provided by algebraic modeling languages.

Feature		**AMPL**	**GAMS**	**JuMP**	**Pyomo**
Modeling	Independent	Yes	Yes	Yes	Yes
	Scripting	Limited	Limited	Yes	Yes
Data	Input	Limited	Limited	Yes	Yes
	Manipulation	No	No	Yes	Yes
Solvers	Total	47	35	14	25
	Global	4	9	2	1
	LP	17	21	9	10
	MCP	1	5	1	1
	MINLP	6	15	3	6
	MIP	14	16	6	8
	MIQCP	5	20	3	4
	NLP	19	17	7	10
	QCP	9	21	6	6
Presolving		Yes	No	No	No
Visualization		No	No	No	No
License	General	Paid	Paid	Free	Free
	Academic	Free	Free	Free	Free

programming languages. The ease of data input for the model differs among AMLs. While all of them support input from a flat file, some more advanced scenarios such as reading data from relational databases are more straightforward in `JuMP`, or `Pyomo`. `AMPL` and `GAMS` require a complicated setup instead (e.g., using ODBC drivers) to access the database. Wherein `JuMP` or `Pyomo`, a standard Julia or Python driver could be used to get data from relational and any other type of database supported by Python or Julia. Manipulation (e.g., transformation) of data is only supported by `JuMP`, and `Pyomo`.

When comparing solver support not only a total number of solvers supported by AML was provided, but also a more granular split based optimization problems types was made. Problem types include: Global optimization, Linear Programming (LP), Mixed Complementarity Problem

(MCP), Mixed Integer Nonlinear Programming (MINLP), Mixed Integer Programming (MIP), Mixed Integer Quadratically Constrained Programming (MIQCP), Nonlinear Programming (NLP), Quadratically Constrained Programming (QCP). Out of all the reviewed AMLs `AMPL` is the one supporting the most of the solvers in total. However, it should be noticed that the categorization of solvers by supported problem types is different among vendors. Thus, in this comparison, information available from vendors was used and harmonized across all of them. Solvers supported by `JuMP` and `Pyomo` require additional explanation. First, both AMLs support solvers compatible with `AMPL` (via AmplNLWriter package or ASL interface). Therefore, any solver that is equipped with an `AMPL` interface can be used by `JuMP` or `Pyomo`. This could allow to state that `JuMP` and `Pyomo` support all `AMPL` solvers. However, solvers supported via the `AMPL` interface were excluded from the list. First of all, for some commercial solvers, it might be needed to request a particular version from the solver's vendor that comes with the `AMPL` interface. So it's not the support out-of-the-box. Second, since `JuMP` and `Pyomo` are open-source, there potentially could be multiple unknown third-party packages adding support for specific solvers for each of the AMLs. So in Table 12 only the solvers mentioned on the official `JuMP` and `Pyomo` websites were counted.

Presolving capabilities are only available in `AMPL`. `JuMP` and `Pyomo` have programming interfaces for creating custom presolvers, however, none of them are provided out of the box. Using Python or Julia libraries, it is possible to visualize the results produced by `Pyomo` and `JuMP`. However, it requires custom development, and none of the standard `JuMP` or `Pyomo` libraries are supporting that.

It is important to conclude that `JuMP` and `Pyomo` are open-source AMLs built on top of general-purpose programming languages, making them fundamentally different from the competitors. This allows researchers familiar with Julia or Python to learn, improve, and use `JuMP` or `Pyomo` much more comfortably. At the same time, it is practically impossible to introduce improvements to commercial counterparts.

2.2.2 Support for parallelism

Parallelism-related features are relatively new in AMLs, and once wisely applied might contribute towards a significant performance increase in solving real-life mathematical optimization problems. Three main use cases of parallelism within algebraic modeling languages can be identified.

First, it is the parallelism in a problem-solving phase implemented by the solver algorithms. There is the opportunity to use parallel computations to aid in the search for global solutions, typically in a non-convex (or discrete) setting. Optimization algorithms also have utilized building blocks, most prominently decomposition and parallel linear algebra techniques, to exploit the computational power of high-performance machines [9].

Secondly, in some applications, optimization of a collection of problems is required, where each problem is structurally the same, but in which some or all data defining the instance is updated [10]. Solving such collections of problems could benefit from a single initiation of the base model instance, updating the base model instance with specific scenario information, and solving the scenarios in parallel.

Lastly, truly large-scale problems may require parallel processing not only for the solution of the problem but also during the model generation phase [13]. For this, an AML that facilitates the modeling of the problem structure and is capable to utilize the problem structure in the parallel model generation is needed.

In this research, focus was placed on the last two types of parallelism within AMLs since the first one is implemented by the solvers and not by the AMLs themselves. In the following sections, a comparison on how prominent AMLs are supporting such types of parallelism is provided.

2.2.2.1 AMPL

`AMPL` itself does not support defining tasks to be executed in parallel [59]. However, there are attempts to extend `AMPL` and provide some features of parallelism. Two most notable examples are `Parampl` [59]. and `SML` [13].

`Parampl` is a Python-based tool for parallel and distributed execution of `AMPL` programs. `Parampl` introduces a mechanism for explicit parallel

execution of subproblems from within the AMPL program code. The mechanism allows dispatching subproblems to separate threads of execution, synchronization of the threads, and coordination of the results in the AMPL program flow, allowing the modeler to define algorithms solving optimization problems with parallel subtasks.

Parampl is a good option if it is needed to solve the same problem with a different set of parameters multiple times. It also allows the practitioner to define complex optimization tasks in a decomposed way. Therefore, the modeler can take advantage of the problem structure and formulate algorithms to solve optimization problems as subtasks. However, the definition of the problem structure is not formalized and problem decomposition is done imperatively.

Parampl is also capable to execute solvers for subproblems on remote machines, thus utilizing shared computing power.

In the benchmark of Parampl [59] authors state that results of the experiments verified that by using distributed Parampl, it was possible to overcome the memory limitation while dispatching subproblems to multiple physical machines. The values were slightly higher for the local execution, since in the distributed mode, a greater portion of time is spent on transferring the problem and solution files to/from the remote machines. In general while solving a Griewank function optimization problem ($n = 4000$) on a cluster with eight dual core machines a speed-up of a factor between 1.9 (using 2 cores) to 7.8 (using 16 cores) was experienced. Speed-up started to degrade once limit of cores was reached.

Structured Modelling Language (SML) is an extension to AMPL that allows the modeler to express the structure inherent to the problem in a natural way. AMPL language is extended by the block keyword that groups together model entities and allows them to be repeated over indexing sets [13]. Since then, models can be constructed from submodels allowing the problem generation phase to be parallelizable too [19].

An SML parser for linear, quadratic, and stochastic programming problems is available. SML also supports integration with OOPS solver (Object-Orientated Parallel Solver) which exploits the model's block structure and can solve subproblems in parallel.

Implementation of the parallel submodel generation process is provided by the authors of PSMG (Parallel Problem Generator for Structure Conveying Modelling Language) [33].

The latest version of SML was released in February 2011, and at the time of writing, the download section on the author's website is not working.

2.2.2.2 GAMS

GAMS has Grid and Multi-Threading Solve Facility [31] which combined with Gather-Update-Solve-Scatter [31] manager can solve multiple scenarios of the same problem in parallel. An example of such a problem definition can be seen in the GUSS Grid model [30] available in the GAMS model library.

The purpose of this new Gather-Update-Solve-Scatter (GUSS) manager is to provide syntax at the GAMS modeling level that makes an instance of a problem that provides limited access to treat that instance as an object, and allows the modeler to update portions of it iteratively [30].

GUSS gathers data from different sources/symbols to define the collection of models (called scenarios), updates a base model instance with this scenario data, solves the updated model instance, and scatters the scenario results to symbols in the GAMS database [10].

Another GAMS tool called GAMS Grid facility allows to take advantage of High-Performance Computing Grids and systems with multiple CPUs. This language feature facilitates the management of asynchronous submission and collection of model solution tasks in a platform-independent fashion [31].

GAMS Multi-Threading Solve Facility allows the asynchronous submission and collection of model solution tasks on a single, multi-threaded machine while using efficient in-memory communication between GAMS and the solver [31].

The multi-Threading feature is in beta status and works in the same way as the GAMS Grid facility. The solve statement generates the model and passes it to the solver in a separate thread, then a handle of the model instance may be stored in the model attribute called handle, and the grid handle function may be used to collect the solution and deal with the model instance.

It is limited to shared memory and is supported only by a limited set of

solvers varying between different platforms.

In 2017, Bussieck, M.and Fiand, F. presented an approach [12] how to reach parallel model generation using existing GAMS tools.

The idea is the following:

1. Use `.stage` attribute as a model annotation to identify specific blocks of the model.
2. Simultaneously using mpirun create n+1 annotated model blocks as GAMS Data eXchange files by using GAMS Convert tool.
3. Use GAMS/PIPS-IPM solver link to solve each of the blocks/sub-problems.

This seems to be a viable option not requiring any new tools to be developed. However, an orchestration-like tool would be a good addition to the proposal, allowing practitioners to easier adopt the method.

Later in 2019, Bussieck, M presented a benchmark of solving large-scale GAMS models on HPC platforms (MPI) [11] using the approach described above. It proved that parallel model generation was able to significantly speed-up process of solving model with tens of millions of variables and constraints. Comparing with other solvers running on multiple threads, the MPI based distributed solver using parallel model generation came at par once starting to use 16 threads (MPI tasks). Speed-up became evident (factor of 2) once running on 16 threads and only steadily increased until 512 threads were used.

Using such MPI based approach also proves that parallel model generation using distributed HPC platforms would help to address Big Data challenges where input/output operations might become a bottleneck.

2.2.2.3 JuMP

As stated by one of the authors [53] of JuMP - parallel model generation is not supported in JuMP out of the box. However, an extension of JuMP called StructJuMP is available [72].

The StructJuMP package provides a parallel algebraic modeling framework for block-structured optimization models in Julia. StructJuMP ,

originally known as StochJuMP, is tailored to two-stage stochastic optimization problems and uses MPI to enable a parallel, distributed memory instantiation of the problem [72]. It is implemented as a front-end for the PIPS-IPM solver.

It can be used to develop models that support both parallel scenario solving and parallel model generation.

2.2.2.4 Pyomo

Pyomo supports distributed computing via the Python Pyro package. However, one of the authors of Pyomo acknowledges that it does not support parallel model generation [36] as such.

The Python package called Pyro provides capabilities that are used to enable progressive hedging (PH) to make use of multiple solver processes for subproblems. This can be utilized to solve multiple scenarios of the same model in parallel.

2.2.2.5 Summary of support for parallelism in AMLs

Summary of tools supporting the parallel model generation and parallel solving of different scenarios of the same model is provided in Table 13.

Type of parallelism	**AMPL**	**GAMS**	**JuMP**	**Pyomo**
Scenario solving	Preampl	GUSS/GRID	StructJuMP	Pyro/PH
Model generation	PSMG	.stage/GDX	StructJuMP	-

Table 13: Support for parallel scenario solving and parallel model generation by the AMLs.

As it can be seen in Table 13 reviewed AML software has limited support for parallelism and all of it comes from nonstandard extensions or composition of existing tools. It is also seen that parallel model generation is in the very early stages of maturity and some authors doubt the benefit of parallel model generation capability as such.

2.3 Conclusions

This chapter described the main differences between the four most prominent algebraic modeling languages: `AMPL`, `GAMS`, `JuMP` and `Pyomo`.

First, AML software were compared considering aspects such as operating system support, licensing, and type of user interface. Next, a comparative analysis of features was conducted, identifying how well AMLs fulfill the requirements for modern AMLs. Furthermore, a deep dive into the promising features supporting parallelism in AMLs was made. Three main use cases of parallelism within algebraic modeling languages were identified, but focus was placed only on the two provided by AMLs and not by solver algorithms. One use case is the optimization of a collection of problems, where each problem is structurally the same, but in which some or all data defining the instance is updated. The second one is large-scale problems requiring parallel processing not only for the solution of the problem but also during the model generation phase.

The following conclusions have been made:

1. All of them support three major operating systems (Windows, Unix, Mac OS), so the usage should be relatively smooth independent of what operating system or hardware modeler is using. Also, all except `Pyomo` provide GUI for writing a textual model code and running some of the standard commands. `AMPL` and `GAMS` being commercial software, has a high minimum (base) cost of 3000 to 4000 USD, once adding the cost of solvers it makes the price rise significantly higher. Trial versions are quite limited (time or model size-wise) and even the academic licensing does not come for free (starting from 400 USD). `Pyomo` and `JuMP` are open source and distributed without a fee, although commercial solvers have to be procured separately.

2. In all reviewed AMLs, the optimization problems are represented in a declarative way. Furthermore, since all of them are part of a specific modeling system, a clear separation between problem definition and the solution process in the context of the modeling system exists. The separation between the problem structure and its data is supported in all

reviewed languages.

3. All reviewed AMLs allow modeling problems in a solver-independent manner. The way of providing data for the model differs among AMLs. While all of them support input from a flat file, some more advanced scenarios such as reading data from relational databases are more straightforward in `JuMP`, or `Pyomo`.

4. When it comes to solver support, `AMPL` is the one supporting the most. Presolving capabilities are only available in `AMPL`. `JuMP` and `Pyomo` only have programming interfaces for creating custom presolvers. Using Python or Julia libraries, it is possible to visualize the results produced by `Pyomo` and `JuMP`. However, it requires custom development, and none of the standard `JuMP` or `Pyomo` libraries are supporting that.

5. Reviewed AMLs have limited support for parallelism and most of it comes from nonstandard extensions or custom orchestration of existing capabilities. Parallel model generation is in the very early stages of maturity and some authors doubt the benefit of parallel model generation capability as such. However, once applied wisely parallelism might contribute towards a significant performance increase in solving real-life mathematical optimization problems.

3 EXPERIMENTAL ANALYSIS OF ALGEBRAIC MODELING LANGUAGES

3.1 Practical comparison of AMLs

For the practical comparison of the selected AMLs, the same classical transportation problem by Dantzig, G. B. [15] introduced in Section 1.3 was chosen.

In this problem, we are given the factories' supplies and the markets' demand for a single commodity. We are also given the unit cost of shipping the product from each factory to each market. The goal is to find the least cost shipping schedule that meets the requirements of all markets and supplies at factories.

The transportation problem formulated as a model in all four considered AML was compared based on the following criteria:

- model size in bytes;
- model size in the number of code lines;
- model size in the number of language primitives used;
- model instance creation time.

Since the transportation problem is a linear programming (LP) type of problem, it was chosen to measure the model instance creation time as the time needed to export a concrete model instance to `MPS`[4] format supported by most LP solvers. The following sources provided sample implementations of the transportation problem for the AMLs under consideration:

- AMPL model in GNU Linear Programming Kit[5]
- GAMS Model Library[6]
- JuMP Examples[7]

[4] http://lpsolve.sourceforge.net/5.5/mps-format.htm
[5] https://github.com/cran/glpk/blob/master/inst/doc/transport.mod
[6] https://www.gams.com/latest/gamslib_ml/libhtml/index.html
[7] https://github.com/jump-dev/JuMP.jl/tree/master/examples

- Pyomo Gallery[8]

Transportation problem models in all different AMLs can be seen either in Listing 2 and Listing 3 or in Appendix A. It is worth noting that for the sake of simplicity, the problem model samples were concrete models, i.e., data of the model instance was described alongside the model structure.

In order to compare models in a uniform way a simplification of the model implementations provided in the literature sources was made in the following way:

- all optional comments, explanatory texts, and documentation were removed;
- all empty lines were excluded;
- parts of the code responsible for calling the solver and displaying results were omitted;
- while counting AML primitives generic functions (*sum*, *for*), data loading directives (*data*), arithmetical and logical operators were excluded.

A comparison of the simplified sample transportation problem model's characteristics in all reviewed AMLs is given in Table 14. The comparison characteristics were chosen due to the following reasons:

- input/output computing operations are the most expensive ones so a dynamic in the size of the model can indicate potential challenges in processing large scale models;
- number of lines of code and numbers of primitives indicate complexity of learning and using AML.

It can be seen from Table 14 that models implemented in `AMPL`, `GAMS`, and `JuMP` are the most compact ones, while the model written in `Pyomo` is more verbose.

While comparing the number of language primitives required to create a model, `JuMP` and `AMPL` showed the best results, which indicates that these

[8] https://github.com/Pyomo/PyomoGallery

Table 14: Comparison of the basic characteristics of simpliefied transportation problem model instances in different AMLs.

Criteria	**AMPL**	**GAMS**	**JuMP**	**Pyomo**
size in bytes	683	652	632	1235
lines of code	24	31	18	29
primitives used	5	8	4	6

modeling languages might have a more gentle learning curve. Therefore, it can be concluded that in the context of the reviewed algebraic modeling languages, `JuMP` and `AMPL` allow formulating an optimization problem most concisely.

The creation time of the transportation problem model instance defined in each AMLs was used to measure a model loading. The process was done in the following steps:

1. loading model instance from a problem definition written in the native AML;
2. exporting model instance to MPS format;
3. measuring total execution time;
4. investigating characteristics of an instance model.

Generated model instances in MPS format can be found in GitHub repository's `models` directory [42].

Table 15: Characteristics of the created transportation model instances.

Characteristic	**AMPL**	**GAMS**	**JuMP**	**Pyomo**
Constraints	6	6	6	6
Non zero elements	13	19	13	13
Variables	7	7	7	7

The characteristics of the created model instances can be seen in Table 15. It can be concluded that all modeling languages have created a model instance

using the same amount of variables and constraints. However, the definition of nonzero elements is different between GAMS and other modeling systems.

Table 16: Total time taken for consecutive transportation model instance creation runs.

No. of runs	**AMPL**	**GAMS**	**JuMP**	**Pyomo**
1 run	30 ms[9]	170 ms	28341 ms	720 ms
10 runs	220 ms	1730 ms	32199 ms	7280 ms
100 runs	2130 ms	16490 ms	58151 ms	79600 ms

In Table 16, the benchmark results of model instance creation time are provided. It was attempted to run multiple consecutive model instance creations (10 runs, 100 runs) to identify if the modeling system uses any caching. It was exhibited that AMPL showed significantly better results compared to others. This allows concluding that AMPL is the most optimized from a performance point of view. On the other hand, the poor JuMP performance confirms Dunning et al. [18] statement that JuMP has a noticeable start-up cost [10] of a few seconds even for the smallest instances. In this case, only the initialization of the JuMP package took around 7 seconds. A significant speed-up in multiple consecutive model instances creation can be seen, which also confirms Dunning et al. [18] results. When a family of models is solved multiple times within a single session, this compilation cost is only paid for the first time that an instance is solved.

3.2 Library of practical optimization problems

GAMS Model Library[11] was chosen as a reference for creating a sample optimization problem suite used for experimental analysis. Automated shell script `gamslib-convert.sh` was created to build such a library. It can be found in the `tools` directory of GitHub repository [42]. A detailed

[9] Single AMPL run takes very little time thus is difficult to measure precisely. Based on 10 and 100 consecutive runs it should be closer to 22 milliseconds.

[10] Start-up cost consists of the precompilation and caching time required to prepare JuMP environment.

[11] https://www.gams.com/latest/gamslib_ml/libhtml/index.html

explanation of how the tool for creating a test library works and issues identified in the GAMS Model Library are provided in the sections below.

3.2.1 Content of the library

Test model library consists of 296 sample problems in AMPL, GAMS, JuMP, and Pyomo scalar model format. It has been built using sample models available in the GAMS Model Library which consists of 423 models some of which had to be excluded due compatibility issues explained in the subsection 3.2.3. An example of the transportation problem in GAMS Model Library converted to GAMS scalar format is shown in Listing 4.

```
Variables  x1,x2,x3,x4,x5,x6,x7;
Positive Variables  x1,x2,x3,x4,x5,x6;
Equations  e1,e2,e3,e4,e5,e6;
e1..  -0.225*x1 - 0.153*x2 - 0.162*x3 - 0.225*x4
        - 0.162*x5 - 0.126*x6 + x7 =E= 0;
e2..  x1 + x2 + x3 =L= 350;
e3..  x4 + x5 + x6 =L= 600;
e4..  x1 + x4 =G= 325;
e5..  x2 + x5 =G= 300;
e6..  x3 + x6 =G= 275;
Model m / all /;
m.limrow=0; m.limcol=0;
Solve m using LP minimizing x7;
```

Listing 4: Transportation problem converted to GAMS scalar model

Library was built by using GAMS CONVERT[12] utility to convert GAMS native models to scalar models for AMPL, GAMS, JuMP and Pyomo algebraic modeling languages.

A full list of available sample models and their characteristics is available in README.md file in the target model library directory.

The list includes links for specific AML models:

- .mod; AMPL scalar model;
- .gms; GAMS scalar model;

[12] https://www.gams.com/latest/docs/S_CONVERT.html

- `.jl`; `JuMP` scalar model;
- `.py`; `Pyomo` scalar model,

and main characteristics of a model:

- number of equations;
- number of variables;
- number of discrete variables;
- number of nonzero elements;
- number of nonlinear nonzero elements.

Each model is placed under a separate directory following the structure:

```
LIBRARY_LOCATION/MODEL_NAME
```

where `LIBRARY_LOCATION` is the directory path provided at generation time and `MODEL_NAME` is the name given to the model in the GAMS Model Library.

Apart from the four scalar models for `AMPL`, `GAMS`, `JuMP`, and `Pyomo`, in each model's directory, the following items exist:

- original `GAMS` model:

  ```
  MODEL_NAME.gms
  ```

- statistics gathered during model conversion:

  ```
  MODEL_NAME-stats.log
  ```

- verbose output of `GAMS CONVERT` tool:

  ```
  MODEL_NAME-scalar.AML_NAME.lst
  ```

- standard output of `GAMS CONVERT` tool:

```
AML_NAME-convert.log
```

where `AML_NAME` is `AMPL`, `GAMS`, `JuMP`, `Pyomo`, and `MODEL_NAME` is the name of the model.

Table 17 provides a fragment of `README.md` file where it can be seen how each model available in the testing library is documented alongside its essential characteristics and direct links to concrete models for the given AMLs.

Table 17: Fragment of README.md file provided in the generated models library directory.

ID	Model	Type	Description	Files	Eq	Vars	Vars Disc	Non zero	Non zero NL
1	trnsport	LP	A Transportation Problem	mod gms py jl	6	7	0	19	0
2	blend	LP	Blending Problem I	mod gms py jl	4	10	0	37	0
3	prodmix	LP	A Production Mix Problem	mod gms py jl	3	5	0	13	0
4	whouse	LP	Simple Warehouse Problem	mod gms py jl	5	13	0	28	0

3.2.2 Building the library

The automated shell script `gamslib-convert.sh` is available in the `tools` directory of GitHub repository [42]. The script was developed to generate the AMLs testing library. The script uses `GAMS CONVERT` tool to convert the `GAMS` proprietary format model to a scalar model in `AMPL`, `GAMS`, `JuMP`, and `Pyomo` formats. Characteristics of the sample problem models (number of equations, variables, discrete variables, nonzero elements, and nonzero nonlinear elements) are automatically extracted and noted. Sample problems are also grouped based on optimization problem types.

Prerequisites to use the script require GAMS Modeling System to be installed on the machine and `gams` and `gamslib` executables to be available in the operating system's `PATH`.

The script provides two execution modes. The first mode is for converting a single model and the second one is for converting all GAMS Library models.

Building a single model:

1. Run gamslib-convert.sh:

   ```
   sh gamslib-convert.sh MODEL_NAME MODEL_TYPE
   ```

   where `MODE_NAME` is `GAMS` model name and `MODEL_TYPE` is type of a particular problem (e.g. LP, MIP, RMIP, QCP, MIQCP, RMIQCP, NLP, DNLP, CNS, MINLP, or MCP).

2. Created model will be placed in `gamslib` directory

Building full library:

1. Create an input CSV file describing models to be converted. Use semicolon as a CSV column separator:

   ```
   MODEL_ID;MODEL_NAME;MODEL_DESCRIPTION;MODEL_TYPE;MODEL_AUTHOR
   ```

2. Run `gamslib-convert.sh`:

   ```
   sh gamslib-convert.sh CSV_FILE
   ```

   where `CSV_FILE` is a file created in step 1), e.g., `gamslib.csv`;

3. Generated scalar models for all models provided in the CSV file will be placed under `gamslib` directory. Errors will be documented in `error.log` file in the `gamslib` directory.

3.2.3 Findings

At the time of writing, there were 423 models in the GAMS Model Library which was used as a base to create test model library used in the experimental analysis. Out of them, it was required to eliminate 66 models using `GAMS`

proprietary modeling techniques (e.g., MPSGE, BCH Facility), 20 using general-purpose programming language features (e.g., cycles), four models tightly coupled to CPLEX and DECIS solvers. It is important to note that 35 models failed to be loaded by a fully licensed GAMS CONVERT tool due to execution or compilation errors. In this meaning, some models in the GAMS Library are not compatible with the GAMS modeling system itself. Thus the resulting test model library resulted having 296 valid models in four different AMLs.

Additionally, while performing the model instance creation benchmark, it was identified that 12 AMPL, 11 JuMP, and 29 Pyomo models generated by the GAMS CONVERT tool had errors in them. Most of the Pyomo errors were caused by an incorrect GAMS CONVERT tool behavior where the definition of the Suffix primitive uses AMPL but not Pyomo semantics. Similar issues were observed in some of the JuMP models. An example of an invalid Suffix primitive generated by GAMS CONVERT and the correct Pyomo syntax for Suffix definition can be seen in Listing 5.

```
# GAMS CONVERT generated Pyomo suffix syntax
suffix ref integer IN;
# Correct Pyomo suffix syntax
ref = Suffix(direction=Suffix.EXPORT, datatype=Suffix.INT)
```

Listing 5: Example of GAMS CONVERT generating an invalid model definition in Pyomo language.

3.3 Benchmarks

All examined AMLs support all types of traditional optimization problems. However, it is unclear how efficiently each AML can handle large model loading and what optimizations are applied during model instance creation. It would also be of great value to analyze how each of the modeling languages performs within an area of a specific type of optimization problem (e.g., linear, quadratic, nonlinear, mixed-integer). To give such a comparison and thoroughly examine the characteristics of AMLs, a more extensive benchmark involving much larger optimization problem models is needed.

Therefore, an extensive library of sample optimization problems for the analyzed AMLs described in previous section has to be used.

3.3.1 Model instance creation time

The generated library was used to determine the amount of time each modeling system requires to create a model instance of a particular problem. A shell script `load-benchmark.sh` was created and is available in the `tools` directory of the GitHub repository [42], which loads each model into a particular modeling system and then exports it to the format understandable by the solvers. It was chosen to use the *.nl* [32] format as the target format acceptable by the solvers, as *.nl* supports a wide range of optimization problem types. The benchmark measures the time the modeling system takes to perform both model instance creation and export operations.

It was chosen to exclude sample problems with conversion errors from the benchmark. More information about errors experienced while generating the testing library is provided in Section 3.2. Thus, only the models that were successfully processed by all modeling systems were compared.This reduced the scope of the benchmark to 268 models.

Benchmark methodology, hardware, and software specifications can be found in the GitHub repository [42]. Detailed results are available in the `model-loading-times.xlsx` workbook in the benchmark section of GitHub repository [42]. A summary of the average model instance creation time split by the problem type is provided in Figure 6.

It can be seen that the trend exhibited in the transportation problem model benchmark persists. `AMPL` is still a definite top performer, while `JuMP` and `Pyomo` perform the worst. There are no significant variations between different optimization problem types except for `JuMP`, where the model instance creation time tends to vary significantly while working with different types of problems. Moreover, as confidence intervals show, the variation between different models of the same type is also more significant once using `JuMP`. Most likely this is caused by Julia's (general purpose programming language used by `JuMP`) dynamic nature and the mix of run time compilation and caching of similar `JuMP` models.

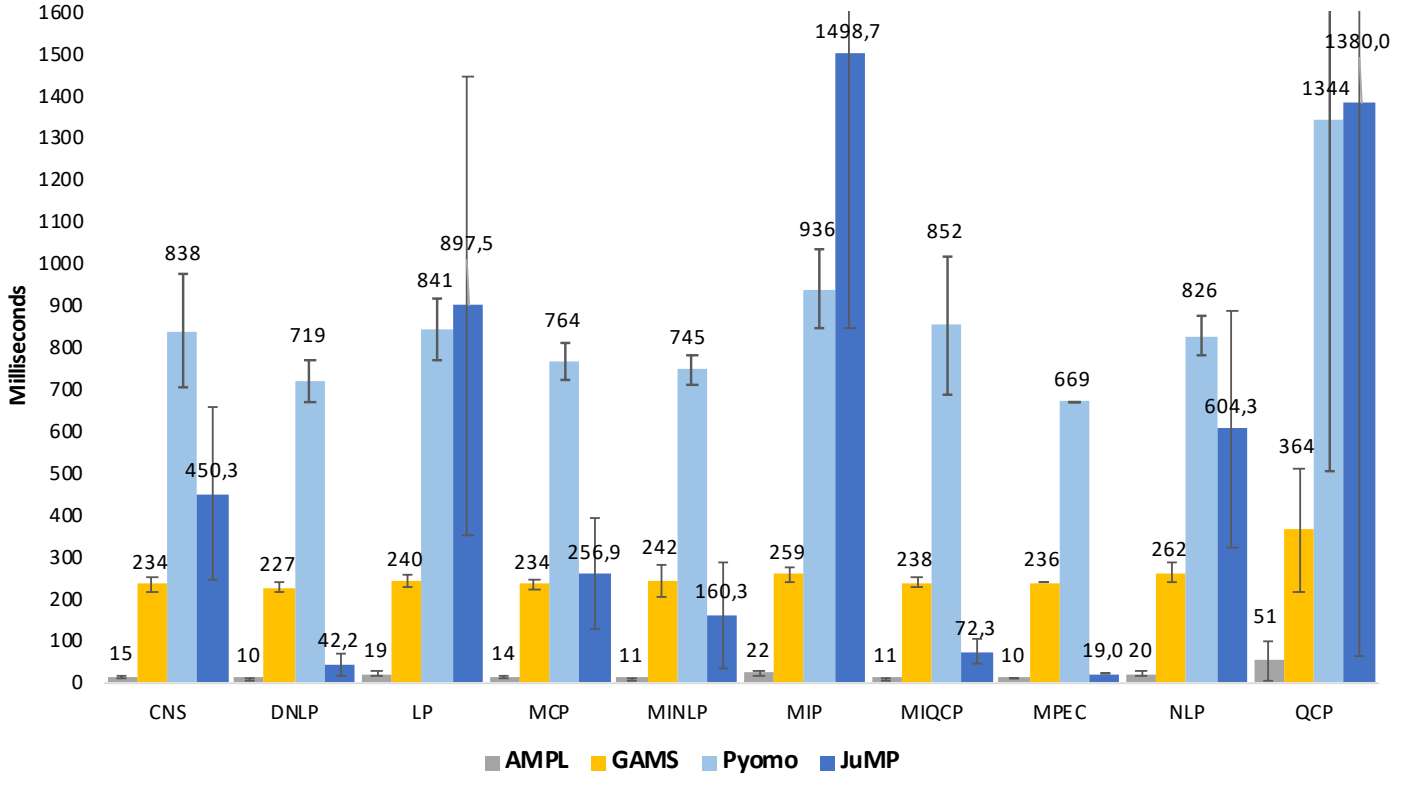

Figure 6: Average model instance creation time in milliseconds. Grouped by optimization problem type.

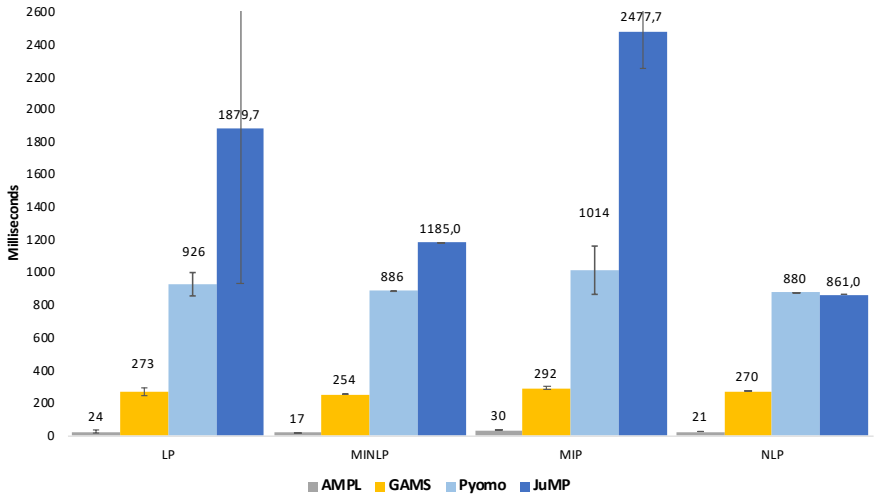

Figure 7: Average large model instance creation time in milliseconds. Grouped by optimization problem type.

It was observed that the average difference between `AMPL` and other contenders increases when the models become larger. Comparing instance creation times of large models (models having more than 500 equations, 8 such models in the testing library) reveals 11 times the difference between `AMPL` and `GAMS`, 38 times the difference between `AMPL` and `Pyomo`, and close to 100 times the difference between `AMPL` and `JuMP`. The difference between `GAMS` and `Pyomo` stayed roughly the same - around 3.5 times. The summary of the large model instance creation time can be seen in Figure 7.

Thus, it can be concluded that out of the reviewed AMLs, `AMPL` is a clear top-performing AML in regard to the model instance creation time.

3.3.2 JuMP benchmark

A similar time benchmark of model instance creation has already been conducted by Dunning et al. [18], where a smaller set of large models is used. While some of the trends exhibited in the benchmark persist (`AMPL` is the fastest, `GAMS` comes second), `JuMP` performance in this and Dunning et al. benchmarks differs significantly. This leads to compare the benchmark methodology and results by conducting the benchmark described by Dunning et al.

First of all, time benchmark methodologies of this research and Dunning et al. differ. In comparison, here it is attempted to be solver independent and instruct AML to export the generated model instance to NL file, Dunning et al. attempt to solve the model using `Gurobi` solver and measure the time until `Gurobi` reports the model characteristics. While approach in this research can be impacted by the system's input/output performance, using a specific solver heavily depends on how the solver interface is implemented for a particular AML.

To understand more, two benchmarks have been conducted - one as described in the original article and the second one using method of exporting to an NL file. Results of the benchmarks can be seen in Tables 18 and 19.

Table 18: JuMP benchmark using Dunning et al. [18] method. Results provided in milliseconds.

Model	**AMPL**	**GAMS**	**Pyomo**	**JuMP (Direct)**	**JuMP (Cache)**
lqcp-500	2093	2271	17000	17388	37317
lqcp-1000	8075	11995	139201	24590	44575
lqcp-1500	18222	38813	322604	39370	66566
lqcp-2000	32615	93586	575406	57597	88833
fac-25	407	480	7442	17517	39245
fac-50	2732	2884	43106	21331	47735
fac-75	9052	12422	150550	31582	57432
fac-100	20998	29144	393200	61326	93129

Before running the benchmarks, it was required to rewrite some parts of the sample `lqcp` and `facility JuMP` models since syntax changes were introduced between JuMP v0.12 (used by Dunning et al.) and JuMP v0.21.5 (used in this research, the latest version at the time of writing). This research's benchmark was also conducted using newer versions of other AMLs - `AMPL` Version 20190207, `GAMS` v.32, `Pyomo` 5.7 (Python 3.8.3), and solver `Gurobi` 9.0.

Additionally, it was intended to test performance of `JuMP's` new abstraction layer for working with solvers called `MathOptInterface.jl`

(MOI). Therefore, it was attempted to try both CachingOptimizer and DIRECT modes. As seen in Table 18, the DIRECT mode performed much better than the CachingOptimizer mode for both lqcp and facility models. An average difference of close to two times in the instance creation time leads us to suggest modelers evaluate MOI type choice based on specific use cases carefully.

Table 19: JuMP benchmark using export to NL method (in milliseconds)

Model	**AMPL**	**GAMS**	**Pyomo**	**JuMP**
lqcp-500	2716	3265	39988	20424
lqcp-1000	10503	14394	161404	80578
lqcp-1500	25402	49822	307121	483268
lqcp-2000	42780	125564	>10 min	>10 min
fac-25	409	502	9420	8163
fac-50	2837	2993	43087	31799
fac-75	10879	13457	143286	219548
fac-100	23474	32128	328170	>10 min

Overall, both benchmarks confirmed observation that JuMP suffers from the long warm-up time required to pre-compile JuMP libraries. Results were also consistent with the patterns exhibited during the full gamslib benchmark performed earlier. It was not possible to reproduce JuMP performance metrics reported by Dunning et al, where JuMP always outperforms Pyomo. Using the original benchmark method, JuMP outperformed Pyomo only once the model size increased. However, while using export to NL file method, JuMP, on the contrary, started to fall behind Pyomo once model size increased.

The reported differences between this research and the original benchmark [18] might be caused by multiple factors such as different JuMP versions used, improved Pyomo performance, or different Gurobi versions. It is important to stress that JuMP is a very actively developed AML that underwent significant changes during the last years. It would be valuable to explore why the performance could have degraded and the reasons for such slow I/O operation performance revealed during writing to an NL file

benchmark.

3.3.3 Presolving benchmark

Another performance-related feature of AMLs is the ability to presolve a problem before providing it to the solver. The presolver can preprocess problems and simplify, i.e., reduce the problem size or determine the unfeasible problem. Only one of the reviewed algebraic modeling languages, AMPL provides presolving capabilities [24].

To assess AMPL's presolving performance, presolving characteristics were gathered while performing the model instance creation time benchmark. For this 286 out of 296 models available in test library were used removing 10 that were converted from GAMS original model to the AMPL scalar model, but had errors and failed to be loaded by AMPL environment.

Table 20: Benchmark of AMPL model presolving. Percentage of an average number of constraints and variables reduced in each model. Grouped by different problem types.

Type	# models	# infeasible	% presolved	% constraints	% variables
CNS	4	0	100.00%	14.63%	31.39%
DNLP	5	0	20.00%	0.00%	7.41%
LP	57	0	36.84%	17.81%	9.66%
MCP	19	0	89.47%	47.00%	8.56%
MINLP	21	1	61.90%	16.32%	9.30%
MIP	61	0	60.66%	19.06%	11.50%
MIQCP	5	2	60.00%	0.00%	2.38%
MPEC	1	0	100.00%	50.00%	0.00%
NLP	101	2	47.52%	9.71%	11.55%
QCP	10	0	60.00%	7.10%	2.55%
RMIQCP	2	0	0.00%	0.00%	0.00%
Total	286	5	52.80%	18.42%	10.73%

A detailed report of the presolving applied to the specific model can be seen in the benchmark section of the GitHub repository [42], while the summary of it can be found in Table 20. It was observed that AMPL presolver managed to simplify the models in 52.8% of the cases, out of which 5 times it

could determine that the problem solution is not feasible, thus not even requiring to call the solver. In total test model library had 7 infeasible models, so AMPL managed to find 70% of them. On average, once applied, the AMPL presolver managed to reduce the model size by removing 18.42% of constraints and 10.73% of variables.

It can be concluded that AMPL presolver is an efficient way to simplify larger problems, leading to improved solution finding performance once invoking a solver with an already reduced problem model instance. Moreover, determining not feasible models can help modelers debug the problem definition process and find errors in the model definition. This allows to argue that presolving is an important capability of any modern AML.

3.3.4 Presolve impact on solving

To evaluate if AMPL presolving has a positive impact on problem-solving, an additional benchmark was conducted. The benchmark included 146 out of 151 models to which AMPL has applied presolve in the model instance creation benchmark. Five models that AMPL presolve determined to be not feasible were excluded from the benchmark. Shell script `solve-benchmark.sh` provided in the tools directory of the GitHub repository [42] was created for executing such a benchmark. The script solves each model using one of the solvers and gathers the output statistics to a report file.

It was chosen to solve the models using Gurobi and BARON solvers. `Gurobi Optimizer (v.8.1.0)` was chosen for solving `LP, MIP, QCP`, and `MIQCP` type of problems. At the same time, `BARON (v.18.11.12)` global solver was chosen for solving `NLP, MINLP, MCP, MPEC, CNS`, and `DNLP` problems. The solvers' choice was motivated by the support for particular problem types[13] and the popularity of solvers based on NEOS Server statistics[14]. Two attempts to solve each model were made. One with AMPL presolver turned on (default setting), and the second one with AMPL presolver turned off. After each run, the solver statistics, including iteration count, solve time (pure solve phase

[13] Gurobi Optimizer Reference Manual: http://www.gurobi.com; BARON User Manual: http://www.minlp.com/downloads/docs/baron%20manual.pdf

[14] https://neos-server.org/neos/report.html

execution time), and objective, were gathered.

It is important to note that both BARON and Gurobi solvers have their presolve mechanisms [66, 2]. Thus, the provided model is simplified by the solver too. This might result in very similar models being solved by the solver despite the AMPL presolve being turned on or off. However, the focus was on estimating AMPL presolve impact in real-life situations; therefore, a full benchmark was executed without changing the default solver behavior. Later on, an additional benchmark was made to estimate the impact of AMPL presolve when solver presolve functionality is turned off.

Detailed AMPL presolve impact on solving report can be found in the GitHub repository's [42] file ampl-solving-times.xlsx sheet Benchmark 1. Table 21 summarizes the positive and negative impact AMPL presolve had on solving the problem iteration and time-wise. Positive impact means fewer iterations or time were needed to solve the problem once the presolve was turned on. A negative impact means the opposite that more iterations or time was required. The impact is considered neutral if the number of iterations did not change or the required time was within the one-second tolerance level.

Table 21: Summary of AMPL presolve impact on solving. Effect iteration and time wise provided.

	Iteration-wise	Time-wise	Iteration-wise (%)	Time-wise (%)
Positive	37	67	26.43%	47.86%
Neutral	74	40	52.86%	28.57%
Negative	29	33	20.71%	23.57%

During this benchmark, 6 models failed to be solved due to solver limitations. Two models were deemed to be not feasible, and two were solved during the AMPL presolve phase. Solvers were capable of solving 41 models during the solver's presolve phase. Moreover, for the six models, the mismatching objective was found with AMPL presolve turned on and off. Overall, AMPL presolve positively impacted 26.43% of the cases iteration-wise and 47.86% time-wise. However, it produced a negative impact in 20.71% of cases iteration-wise and 23.57% time-wise.

As mentioned earlier, both BARON and Gurobi solvers have their presolve mechanisms. An additional benchmark was made to test the impact of AMPL presolve with disabled solver presolving. Since only Gurobi allows the user to disable presolve functionality, a subset of models previously solved with Gurobi was chosen. Detailed benchmark results can be seen in the GitHub repository's [42] file ampl-solving-times.xlsx sheet Benchmark 2. The summary of the benchmark is provided in Tables 22 and 23. Gurobi could not solve two MIP problems (clad and mws) in a reasonable time once Gurobi's presolve functionality was turned off. Those models were excluded from the benchmark.

Table 22: AMPL presolve impact with Gurobi presolve on

	Iteration-wise	Time-wise	Iteration-wise (%)	Time-wise (%)
Positive	18	39	28.57%	61.90%
Neutral	34	0	53.97%	0.00%
Negative	11	24	17.46%	38.10%

Table 23: AMPL presolve impact with Gurobi presolve off

	Iteration-wise	Time-wise	Iteration-wise (%)	Time-wise (%)
Positive	33	44	54.10%	72.13%
Neutral	10	0	16.39%	0.00%
Negative	18	17	29.51%	27.87%

As seen once comparing these results in Table 22 and Table 23, the AMPL presolve had a greater positive effect both iteration-wise (+25.5%) and time-wise (+10.2%) once Gurobi presolve was turned off. AMPL presolve also had a less neutral impact once the solver presolving was off, thus leading to the conclusion that during the first benchmark, some models were simplified to very similar ones before actually solving them.

As it can be seen from the benchmarks, the presolving done by AML has inconclusive effects on the actual problem solving both iterations and time-wise. However, a positive impact is always more significant than the

negative one, and it especially becomes evident once the solver does not have or use its problem presolving mechanisms. This allows to conclude that the presolving capability of AML is an important feature of a modern algebraic modeling language. It can also advised to choose AML having presolving capabilities in case the solver used to solve the problem does not have its presolving mechanism.

3.4 Summary of findings

From the research, it can be concluded that `AMPL` allows us to formulate an optimization problem in the shortest and potentially easiest way while also providing the best performance in model instance loading times. It also leverages the power of model presolving, which helps the modelers in both problem definition and efficient solution finding processes. `GAMS` is a powerful runner-up providing very similar to `AMPL` problem formulation capabilities although running behind in the model instance creation time. Open-source alternatives `JuMP` and `Pyomo` are on par with commercial competitors in the problem definition process. However, the performance of model instance creation is a bit behind compared to its competitors. `JuMP` suffers from noticeable environment start-up costs, while `Pyomo` performance tends to downgrade once the model's size increases.

3.5 Conclusions

In this chapter, multiple different experimental analyses of the chosen AMLs and their findings were presented. Also content of the testing library for practical optimization problems developed during the research was presented.

First, the classical transportation problem modeled with different AMLs was examined. The comparison based on the following criteria was made: model size in bytes, model size in the number of code lines, model size in the number of language primitives used, model instance creation time.

Next, the characteristics and metrics of the models existing in the testing library provided were presented. The library consists of 296 sample problems in `AMPL`, `GAMS`, `JuMP`, and `Pyomo` scalar model format. The library has been built using sample models available in the GAMS Model Library. Near each

model, the type of optimization problem, number of equations, variables, discrete variables, nonzero elements, and nonlinear nonzero elements were described.

Following, the tools and processes for building the testing model library from scratch were explained. The most important tool developed is an automated shell script `gamslib-convert.sh` able to (re)generate the AMLs testing library. The tool is freely available on the GitHub repository. It not only converts given `GAMS` models to other AMLs, but also extracts and documents all model characteristics identified during the conversion process. It also supports two different execution modes - bulk generation of all model library and generating a single model, which might come in handy for debugging purposes

Next, the results of a large-scale model instance creation benchmark were presented. It measured the time the modeling system takes to perform both model instance creation and export operations. This time a benchmark against all models available in the testing library was made. Also, `load-benchmark.sh` shell script, which loads each model into a particular modeling system, then exports it to the format understandable by the solvers, captures execution statistics, and generates a benchmark report, was developed and was made available on the GitHub repository. The tool is freely available for other researchers to use in any benchmark of model instance creation times.

Following, a comparison between `JuMP` benchmark made during this research and the one conducted by Dunning et al. was made.

And finally, a benchmark assessing presolving in `AMPL` was performed trying to identify what benefits for solving the problem it brings.

The following conclusions have been made:

1. Models implemented in `AMPL`, `GAMS`, and `JuMP` are the most compact ones, while the models written in `Pyomo` are more verbose. Comparing the number of language primitives required to create a model, `JuMP` and `AMPL` showed the best results. This could indicate that these modeling languages might have a more gentle learning curve, also allowing to conclude that in the context of the reviewed algebraic

modeling languages, JuMP and AMPL enable practitioners to formulate an optimization problem most concisely.

2. Based on transportation model instance creation time it was experienced that AMPL is standing out from other AMLs and is the most optimized from a performance point of view. Also poor performance of JuMP confirmed Dunning et al. statements that JuMP has a noticeable start-up cost.

3. Building test model library revealed that 35 models failed to be loaded by a fully licensed GAMS CONVERT tool due to execution or compilation errors. Meaning that some models in the GAMS Library are not compatible with the GAMS modeling system itself. Later while performing the model instance creation benchmark, it was identified that 12 AMPL, 11 JuMP, and 29 Pyomo models generated by the GAMS CONVERT tool had errors in them. Most of the Pyomo and JuMP errors were caused by an incorrect GAMS CONVERT tool behavior once creating the definition of the Suffix primitive.

4. The same trend as in the transportation problem model benchmark was exhibited during full benchmark of testing library. AMPL was still a definite top performer, while JuMP and Pyomo performed the worst. It was also observed that the average difference between AMPL and other contenders increase when the models become larger. There are no significant variations between different optimization problem types except for JuMP, where the model instance creation time tends to vary significantly while working with different types of problems. Moreover, the variation between different models of the same type is also more significant once using JuMP. This might be caused by Julia's dynamic nature, resulting in the mix of run time compilation and caching of similar JuMP models. It can be concluded that out of the reviewed AMLs, AMPL is a clear top-performing AML on the model instance creation time.

5. Conducted JuMP benchmark confirmed the observation that JuMP suffers from the long warm-up time required to pre-compile JuMP

libraries. It was not possible to reproduce the JuMP performance metrics reported by Dunning et al., where JuMP always outperforms Pyomo. The reported differences between benchmark in this research and the original benchmark might be caused by different JuMP versions used, improved Pyomo performance, or different Gurobi solver versions.

6. AMPL presolver managed to simplify the models in 52.8% of the cases, out of which 5 times it could determine that the problem solution is not feasible, thus not even requiring to call the solver. In total there were 7 infeasible models in the test library. On average, once applied, the AMPL presolver managed to reduce the model size by removing 18.42% of constraints and 10.73% of variables. Overall, AMPL presolve positively impacted 26.43% of the cases iteration-wise and 47.86% time-wise. However, it produced a negative impact in 20.71% of cases iteration-wise and 23.57% time-wise. It can still be concluded that positive impact is always more significant than negative one, and it becomes evident once the solver does not have or use its problem presolving mechanisms.

4 DIFFERENCES AND SHORTCOMINGS OF ALGEBRAIC MODELING LANGUAGES

In the previous sections, the essential characteristics of modern algebraic modeling languages were explored and it was compared how each of the most prominent AMLs matches those characteristics in theory and practice.

In this section, reproducibility of research results is discussed and results are examined to explore `AMPL`, `GAMS`, `JuMP`, and `Pyomo`, in order to identify differences, which makes the most prominent AMLs difficult for practitioners to learn and use. Each of the highlight characteristics are taken one by one and differences and shortcomings are described.

4.1 Reproducibility of results

Reproducibility is a cornerstone of science [52], and most research fields are affected by reproducibility crisis [20]. Criticism to the current scientific publication process include concerns about fairness, quality, performance, cost, and accuracy of the evaluation processes [40]. Recently, surprising results have been published in the prestigious Nature and Science journals where it was revealed that as many as 77% of biologists and 87% of chemists failed to reproduce experiments published by other scientists [6]. Even more worrying more than half of scientists failed to replicate their own experiments. This leads to new suggestions for improving the reproducibility of studies by adjusting research methods and publication protocols [48]. Within the field of Operations Research difficulties of reproducing many key results also exist [7]. It can happen due to the disconnection between publications and used codes/models, underlying data, parameter settings, etc., as they lack critical details. Moreover, re-execution of computational experiments often requires extensive computing resources, specialized hardware, and/or massive data utilization, which contradicts the sustainable development paradigm.

In this research variety of measures have been carried out to ensure that results can be reproduced. This includes setting up the environment to run benchmarks, tooling to run benchmarks and measure results, and the way of

storing gathered results.

First, it was chosen to use GitHub as a repository for storing and versioning all research related artifacts. All AML benchmark related information is stored in a single repository[15] which includes: a) results of the benchmarks b) library of models the benchmarks were run against c) tooling for running the benchmarks and generating the model library. Each of the sections in the repository have own documentation where key characteristics and knowledge is captured.

Within benchmark section the setup of benchmark environment is clearly defined including hardware and software specifications. Also, a brief description of benchmark methodology and findings is provided. This not only includes results, but also issues faced and reasoning why those happened. Benchmark results are provided not only in textual format but also as plots using confidence intervals. This also allows reproducing results within a given certainty.

In the tooling section automated shell scripts are made available for running the benchmarks or generating models' library. Scripts can be used by anyone with minimal understanding of what they do to reproduce the benchmark using the original methodology. Additionally, the behaviour and logic of the scripts is described, allowing adjusting the scripts in case interfaces of given AMLs or other tools change.

Lastly, library of sample models used for the benchmark is provided, this way ensuring that identical models could be taken and used for reproducing original results of the benchmark. Also, characteristics of each model is described in details, thus allowing to compare it with newer versions if such would be made available.

All the described measures allows to believe that this research should not suffer from the reproducibility crisis and results could be reproduced by both author of the research, but also any other researcher having access to the code repository of the research.

[15] https://github.com/vaidasj/alg-mod-rev

4.2 Features and compatibility

Observing the basic features of AMLs provided in Table 9 it can be noticed that none of them have a full-fledged graphical user interface (GUI) to do the modeling.

AMPL and GAMS provide a simple GUI for writing a textual model code and running standard commands. JuMP and Pyomo do not have any graphical interface at all.

They all support three major operating systems (Windows, Unix, and Mac). Therefore, the usage is relatively the same independent of what the operating system or hardware modeler is using. However, it requires local installation, and thus access to the same physical machine is needed.

AMPL and GAMS are commercial tools with academic licenses starting at USD 500 and the basic commercial license from USD 4000. However, adding more solvers might easily double the price. JuMP and Pyomo are open source and distributed for free, although the solvers have to be procured separately.

The syntax to describe the problems for different AMLs is a noticeable difference to explore. Fragniere and Gondzio [27] state that the algebraic design approach used in AMLs should allow practitioners without specific programming or modeling knowledge to be efficient in describing the problems to be solved.

However, observing constraints of the same problem defined in Listing 6, it can be concluded that it does require some programming knowledge to use AMLs such as JuMP and Pyomo, and it is also not a straightforward process to switch between different AMLs. Exploring more complex language structures (e.g., calculated parameters, suffixes) showcases even more differences between the syntax of AMLs and complexity for practitioners to learn multiple of them.

```
# AMPL
minimize cost: sum{i in I, j in J} c[i,j] * x[i,j];
s.t. supply{i in I}: sum{j in J} x[i,j] <= a[i];
s.t. demand{j in J}: sum{i in I} x[i,j] >= b[j];

# GAMS
cost..      z =e= sum((i,j), c(i,j)*x(i,j));
supply(i).. sum(j, x(i,j)) =l= a(i);
demand(j).. sum(i, x(i,j)) =g= b(j);

# Pyomo
def objective_rule(model):
  return sum(model.c[i,j]*model.x[i,j] for i in model.i for j in model.j)
model.objective = Objective(rule=objective_rule, sense=minimize)
def supply_rule(model, i):
  return sum(model.x[i,j] for j in model.j) <= model.a[i]
model.supply = Constraint(model.i, rule=supply_rule)
def demand_rule(model, j):
  return sum(model.x[i,j] for i in model.i) >= model.b[j]
model.demand = Constraint(model.j, rule=demand_rule)

# JuMP
@objective(model, Min, sum(cost_f[i, j] * trans[i, j]
        for i in 1:length(ORIG), j in 1:length(DEST)))
@constraint(model, [i in 1:length(ORIG)],
        sum(trans[i, j] for j in 1:length(DEST)) <= supply[i])
@constraint(model, [j in 1:length(DEST)],
        sum(trans[i, j] for i in 1:length(ORIG)) >= demand[j])
```

Listing 6: The objective function and constraints of transportation problem [15] expressed in AMPL, GAMS, Pyomo, and JuMP syntax.

The compatibility between AMLs from a tooling perspective is also scarce. `GAMS CONVERT` [28] is the only tool (also commercial) capable of converting between different AMLs. However, the conversion results in a scalar model being produced where the original model structure is lost. An example of a `GAMS CONVERT` generated scalar model in `JuMP` format can be seen in Listing 7. Once compared to the model in an original `JuMP` format, as seen in Listing 3, the scalar model becomes more difficult to read, understand, and extend.

```
using JuMP, MathOptInterface
model = m = Model()

@variable(m, 0 <= x1, start=0)
@variable(m, 0 <= x2, start=0)
@variable(m, 0 <= x3, start=0)
@variable(m, 0 <= x4, start=0)
@variable(m, 0 <= x5, start=0)
@variable(m, 0 <= x6, start=0)

@objective(m, Min, 0.225*x1 + 0.153*x2 + 0.162*x3 + 0.225*x4
                   + 0.162*x5 + 0.126*x6)
@constraint(m, x1 + x2 + x3 <= 350)
@constraint(m, x4 + x5 + x6 <= 600)
@constraint(m, x1 + x4 >= 325)
@constraint(m, x2 + x5 >= 300)
@constraint(m, x3 + x6 >= 275)
```

Listing 7: The classical transportation problem [15] expressed in a JuMP scalar format.

Furthermore, during the benchmarks, a few flaws of the `GAMS COVERT` tool were identified. Around 3% of models available in the GAMS library were converted to `AMPL`, `JuMP`, and `Pyomo` with syntax errors, making them unsolvable. Most of the `Pyomo` errors were caused by incorrect `GAMS COVERT` tool behavior where the definition of the `Suffix` primitive uses `AMPL` but not `Pyomo` semantics. Similar issues were observed in some of the `JuMP` models.

4.3 Solvers

Solvers are an essential part of what a modern AML offers. They implement appropriate solution algorithms to solve the problem at hand. Some solvers are distributed together with AMLs, while others can be purchased separately.

Since the "No Free Lunch Theorems for Optimization" [77] states that for certain types of mathematical problems, the computational cost of finding a solution, averaged over all problems in the class, is the same for any solution method. Thus, no single universal solver for all problem types can exist. This is why in Table 24, an overview of the solvers supported by different AMLs

grouped by problem types is provided. As solvers usually support several types of optimization problems, the last row reflects the total number of unique solvers.

Table 24: Number of solvers supported by AMLs grouped by problem type.

Type	**AMPL**	**GAMS**	**JuMP**	**Pyomo**
Global	4	9	2	1
LP	17	21	9	10
MCP	1	5	1	1
MINLP	6	15	3	6
MIP	14	16	6	8
MIQCP	5	20	3	4
NLP	19	17	7	10
QCP	9	21	6	6
Total	47	35	14	25

The quality of the algorithms implemented by the solvers is also important. In this research, `AMPL` and `GAMS` were identified as the ones providing the most extensive set of state-of-the-art solvers for various types of mathematical optimization problems. The list of supported `AMPL`[16] and `GAMS`[17] solvers is continuously updated and growing . Some solvers might be more significant since having support for multiple problem types makes it appealing for practitioners. However, being universal does not guarantee the best performance. Thus, practitioners might prefer more specialized but also more efficient solvers.

It can be observed that `AMPL` is the one supporting most solvers. AMPL also comes with most solvers bundled in a standard package. However, it should not be confused that the solvers supported by `AMPL` are the same solvers supported by other AMLs. It is possible to get into a situation where a specific problem needs a solver, which is only available through one and only AML.

[16] AMPL Solvers: https://ampl.com/products/solvers/all-solvers-for-ampl

[17] GAMS Solvers: https://www.gams.com/latest/docs/S_MAIN.html#SOLVERS_MODEL_TYPES

This would require the modeler to define the model in a given AML, and it must be known upfront. This is not always the case in real-life situations where the nature of the problem is not fully understood until it has been defined in a specific AML and attempted to be solved.

4.4 Performance

Performance becomes essential once in the need to solve complex real-life problems. Models tend to grow in size and input data amount. Thus, there is a need to focus on the solvers' performances, i.e., solution time, and consider the potential savings in a model instance generation phase. Benchmark results in Figures 6 and 7 show significant model instance generation time variation between different AMLs. It is important to note that while `AMPL` is a clear top performer, open-source counterparts have varying results within different problem types. Thus, choosing the right AML for a concrete problem type would impact performance.

Presolving feature supported by very few AMLs attempts to reduce the problem size or determine the problem to be unfeasible even before sending it to the solver. Out of the benchmarked AMLs, only `AMPL` supports presolving. In this research it was observed that an `AMPL` presolver managed to simplify models in 52.8% of the cases, out of which five times it determined that the problem solution is not feasible, thus not requiring to call the solver. On average, once applied, the `AMPL` presolver managed to reduce the model size by removing 18.42% of constraints and 10.73% of variables.

The benchmark seen in Table 21 was made to test the impact of the `AMPL` presolve to solution time. The results were positive, allowing us to conclude that the presolver is an efficient way to simplify larger problems leading to improved solution finding performance once invoking a solver on an already reduced problem model instance. Moreover, the ability to determine infeasible models can help modelers in the problem definition process debug and find errors in the model definition.

Parallelism is a significant feature for solving real-life mathematical optimization problems. Three prominent use cases of parallelism within AMLs can be identified.

First, it is the parallelism in a problem-solving phase implemented by the solver algorithms. There is the opportunity to use parallel computations to aid in the search for (global) solutions, typically in a nonconvex (or discrete) setting. Mathematical optimization algorithms have also utilized building blocks, most prominently decomposition and parallel linear algebra techniques, to exploit the computational power of high-performance machines [9].

Secondly, in some applications, optimization of a collection of problems is required where each problem is structurally the same. Still, some or all data defining the instance is updated [10]. Solving such collections of problems could benefit from the single initiation of the base model instance, updating the base model instance with specific scenario information, and solving the scenarios in parallel.

Lastly, truly large-scale problems may require parallel processing for the solution of the problem and during the model generation phase [13]. For this, it is needed to have an AML that facilitates the modeling of the problem structure and can utilize the problem structure in the parallel model generation.

This research focuses on the last two types of parallelism within AMLs since the first one is implemented by the solvers and not by the AMLs themselves.

Table 13 provides a brief overview of how specific AMLs can implement the two types of parallelism within the interest of this research.

It is worth noting that in none of the AMLs, parallel scenario solving or parallel model generation is implemented by default. The tools or techniques cited in Table 13 are provided by the scientific society, not the vendors themselves.

It can be concluded that AMLs have limited support for parallelism and all of it comes from nonstandard extensions or composition of existing tools. Thus, parallelism is an important feature of AML and the efforts by the scientific society to address the lack of it serve as proof.

4.5 Summary of findings

Summarizing all observations about differences within AMLs described so far, finding are the following:

- Practitioners must learn the specific syntax of a given AML, which is coupled to a specific modeling environment. Practitioners are not flexible to reuse the knowledge and simplify work;
- Very limited cross-compatibility between different AMLs makes it practically impossible to transfer the model from one AML to another automatically. This might result in the vendor lock-in choosing to stay with a specific AML due to the increased cost of switching;
- Different AMLs support different solvers, so, in some scenarios, practitioners might have a solver available in another AML than the model is written and will not be able to utilize it;
- Different AMLs have different levels of support for various model types. Since in the beginning, it is not always clear what type of problem is being dealt with, it is risky to choose AML, which might not be supported;
- As identified in the practical benchmark, AML performance differs significantly between modeling environments and model types. It is beneficial to be flexible in choosing the best one for large models;
- Varying support for additional capabilities such as presolving or parallel solving.

This leads to believe that while there are a few powerful modeling environments and AMLs, neither provide a complete feature set required for the efficient and intuitive modeling of mathematical optimization problems. Difficulty to switch between AMLs and the need to learn the specific syntax can become a challenge in teaching mathematical optimization in schools and universities.

4.6 Conclusions

In this chapter the main differences and shortcomings among the most prominent AMLs were identified during the research work.

First, general features, syntax, and compatibility challenges were showcased. Next, performance differences and more sophisticated features such as parallelism were presented. Finally, a summary of them findings is provided.

The following conclusions have been made:

1. It was noted that it is rather complicated to reuse the knowledge and simplify work when switching between AMLs, leading to potential vendor lock-in, i.e., the need to stay with a specific AML due to the increased cost of switching. Moreover, since in the beginning, it is not always clear what type of problem one is dealing with, practitioners might choose a wrong AML not supporting a given problem type or end up having a solver available only in another AML than the model is written and not being able to utilize it.

2. AML performance differs significantly between modeling environments and model types. It is beneficial to be flexible in choosing the best one for large models or in those cases when differently supported features such as presolving or parallelism are required.

3. While there are only few powerful modeling environments and AMLs, neither provide a complete feature set which is required for the efficient and intuitive modeling of mathematical optimization problems. The difficulty to switch between AMLs and the need to learn the specific syntax can become a challenge in teaching mathematical optimization in schools and universities.

5 UNIVERSAL OPTIMIZATION SYSTEM

In this research, the major differences and shortcomings among four prominent algebraic modeling languages were identified. To address the identified gaps, a concept of a "universal" optimization system is proposed. It should be a user-friendly environment supporting most of the prominent AMLs and suitable for both an experienced practitioner, but also a student starting to learn mathematical optimization.

5.1 Key concepts of the universal optimization system

First of all, it is suggested to take an open-source and web-based approach to make it much more accessible for a wider audience of users. The web-based approach does not require local installation, thus also opens up for scaling opportunities and utilization of cloud computing power.

Next, the following requirements are set for such a system:

1. It should not require any previous syntax knowledge of any specific AML.
2. Every user action should be done via a guided graphical user interface.
3. System should internally combine and utilize the best characteristics of different AMLs.
4. System should be capable of converting models between different AMLs.
5. It should support all of the solvers available in the underlying AMLs, thus providing the widest range of free and commercial solvers.
6. It should enable presolving and parallel solving features.
7. System should provide a framework for other contributors to extend it. E.g., allowing to add a feature for choosing the best suitable solver for a given model type automatically.

Such a universal optimization system requires two main building blocks to be defined and developed. First, it is a formal language capable of

capturing problem characteristics (called WebAML). The second one is a software system allowing to construct the model using WebAML language and solve it using the underlying AMLs. In the following sections, both concepts are presented in more details.

5.2 WebAML language

First, a proposal for a new formalized algebraic modeling language called WebAML is made.

Once deciding on how to design and define WebAML language for structuring, validating, and capturing the mathematical optimization model logic, the following criteria has been taken into account:

- Models will be built using a graphical user interface. Thus, there is no need to have a short and straightforward syntax;
- Model has to be strictly typed and well structured to allow converting it to the syntax of other AMLs;
- Model will be used on the Web utilizing web browser and HTTP protocol. Thus, the data interchange format should be lightweight, open, standardized, and well adopted on the Web.

Based on the criteria above, it was decided to choose JSON [8] as a lightweight data-interchange format and JSON Schema [79] as a metadata format to describe and validate WebAML data format. Most programming languages widely support JSON, which is human-readable, and have a small metadata footprint compared to other formalized formats such as, e.g., XML. Using JSON Schema creates additional value since tooling to generate OpenAPI[18] based services from JSON Schema can simplify the development of a prototype while also supporting easier improvements once the WebAML language format evolves.

There was also a need to choose how to capture the mathematical equations needed to define constraints in the model. It was chosen to use a

[18]The OpenAPI: language-agnostic interface to RESTful APIs; https://swagger.io/specification

well-known language as LaTeX instead of less adopted counterparts such as ASCII Math[19]. This was dictated by both widespread knowledge of LaTeX in academic society, and the existence of libraries capable of tokenizing LaTeX formulas[20] and displaying them nicely on the Web using W3C MathML [57] standard.

A detailed structure of a WebAML model is defined in JSON Schema file called `webaml.schema.json`, available on the GitHub repository. At the same time, a summary of the basic components is provided in Table 25.

Table 25: Basic components of WebAML language.

Component	**Type**	**Comment**
Set	Single-dimensional	String and number data types supported
Table	Two-dimensional	Number data type supported
Parameter	Scalar	Fixed value
	Indexed	Can be calculated
Variable	Continuous	Upper/lower bound supported
	Binary	-
	Integer	Upper/lower bound supported
Constraint	Simple	For a single variable
	Indexed	Defined over a set
Objective	Minimize	Single objective only
	Maximize	Single objective only

As seen in Table 25, at this moment WebAML language supports only the basic features. Thus, no syntactic sugar is maintained (e.g., aliases), and only single-dimensional sets and two-dimensional arrays are supported. Features such as indexing over partial sets are also not supported. However, WebAML JSON Schema is defined flexibly. If the tooling working with the WebAML language would support more features, it would be easy to extend the WebAML language by introducing new enumerated type values to the schema definition.

[19] AsciiMath: an easy-to-write markup language for mathematics; http://asciimath.org
[20] LaTeX.js: JavaScript LaTeX to HTML5 translator; https://latex.js.org

Listing 8 demonstrates how the same constraints of the transportation problem provided in Listing 6 look in WebAML syntax. While being more verbose, it can be noticed that it is also much more structured, thus making it easier to write code capable of interpreting and converting it to different formats.

```
"constraints": [
  {
          "name": "supply",
          "type": "INDEXED",
          "indexes": ["i"],
          "value": "\\sum_j x_{ij} \\leq a_i",
          "description": "Observe supply limit at plant"
  },
  {
          "name": "demand",
          "type": "INDEXED",
          "indexes": ["j"],
          "value": "\\sum_i x_{ij} \\geq b_j",
          "description": "Satisfy demand at market"
  }
],
"objectives": [
  {
          "name": "cost",
          "type": "MINIMIZE",
          "value": "\\sum_i \\sum_j c_{ij} * x_{ij}",
          "description": "MINIMIZE transportation cost"
  }
]
```

Listing 8: Constraints of transportation problem [15] expressed in WebAML syntax.

5.3 Prototype of the universal optimization system

The second building block supporting the vision of the universal optimization system is a tool allowing to construct the model using WebAML language and solve it using the underlying AMLs.

In the scope of this research, it was decided to build a prototype of such a tool limiting its features to three key operations: 1) load the model from a file 2) solve the model, and 3) export the model to a file. More advanced features such as presolving or parallel model generation, were consciously scoped out, focusing on proving the viability of such a universal optimization system. However, in the prototype clear extension points were designed, allowing for future improvements and adding new features.

All results and code base are available in the WebAML GitHub repository [43], which is structured as follows:

- `webaml-schema` directory contains WebAML language definitions and transportation problem example;
- `webaml-c4model` directory contains an architecture model for a reference implementation of the universal optimization system;
- `webaml-backend` directory contains Java-based back-end services of the prototype;
- `webaml-frontend` directory contains a prototype of a single page front-end application for building and manipulating WebAML models.

As identified in earlier findings, the aim was to build an extendable, open-source, web-based prototype combining the best characteristics of the underlying AMLs. Figure 8, provides the system architecture landscape viewpoint for this prototype of the universal WebAML Optimization System using C4 architecture model[21] notation. A more detailed system component diagram can be seen in Appendix B Figure 12. All of the code is provided on the WebAML GitHub repository [43].

It can be observed that the WebAML modeling and mathematical optimization tool acts as an orchestrator that allows the AML modeler to build the model, convert it to a specific AML format, and send it for solving via local solvers or remote solving in the NEOS Server platform. This way, no new solving capabilities are built, but instead it is building on top of the best features provided by the most prominent AMLs.

[21] The C4 model for visualizing the software architecture; https://c4model.com

It also was proposed to utilize the `GAMS CONVERT` tool to support an even wider variety of AMLs. Adding the support for a new AML that `GAMS CONVERT` already supports is as easy as converting WebAML to `GAMS` (which already has native support in the prototype) and then using `GAMS CONVERT` to convert to any other target AML.

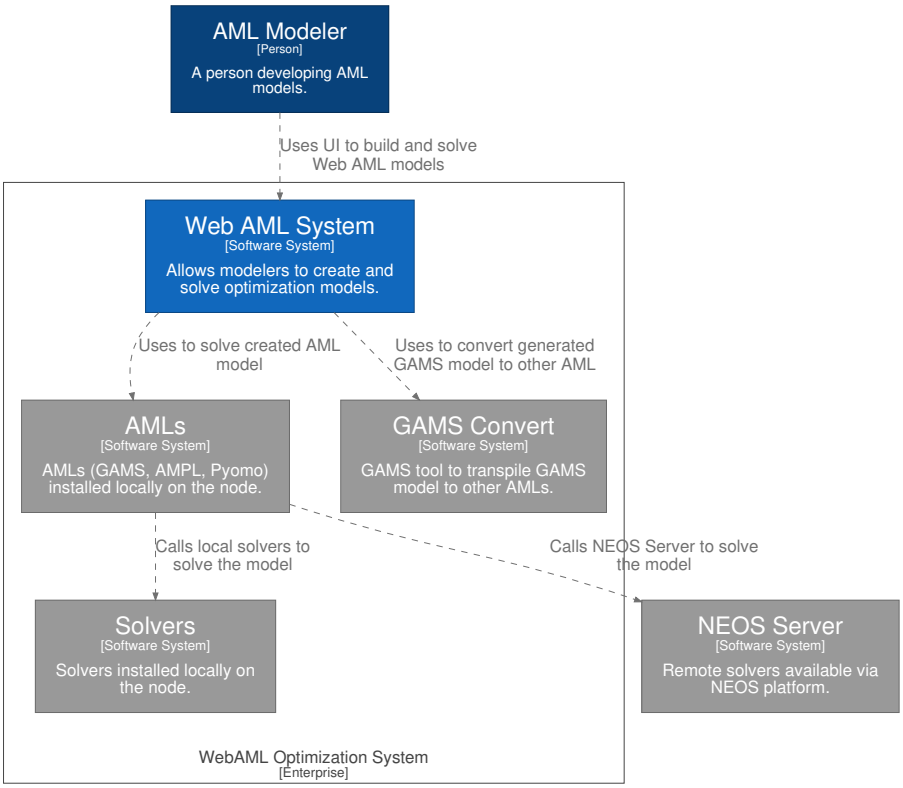

Figure 8: C4 architectural model of a system landscape for the prototype of WebAML tool. Grey boxes indicate external elements provided by other systems.

Herein it was shown how such an approach can work in the `AMPL` and `Pyomo` converters provided in the prototype. A detailed description of how such a solution works while using `AMPL` as an underlying AML is provided in Figure 9.

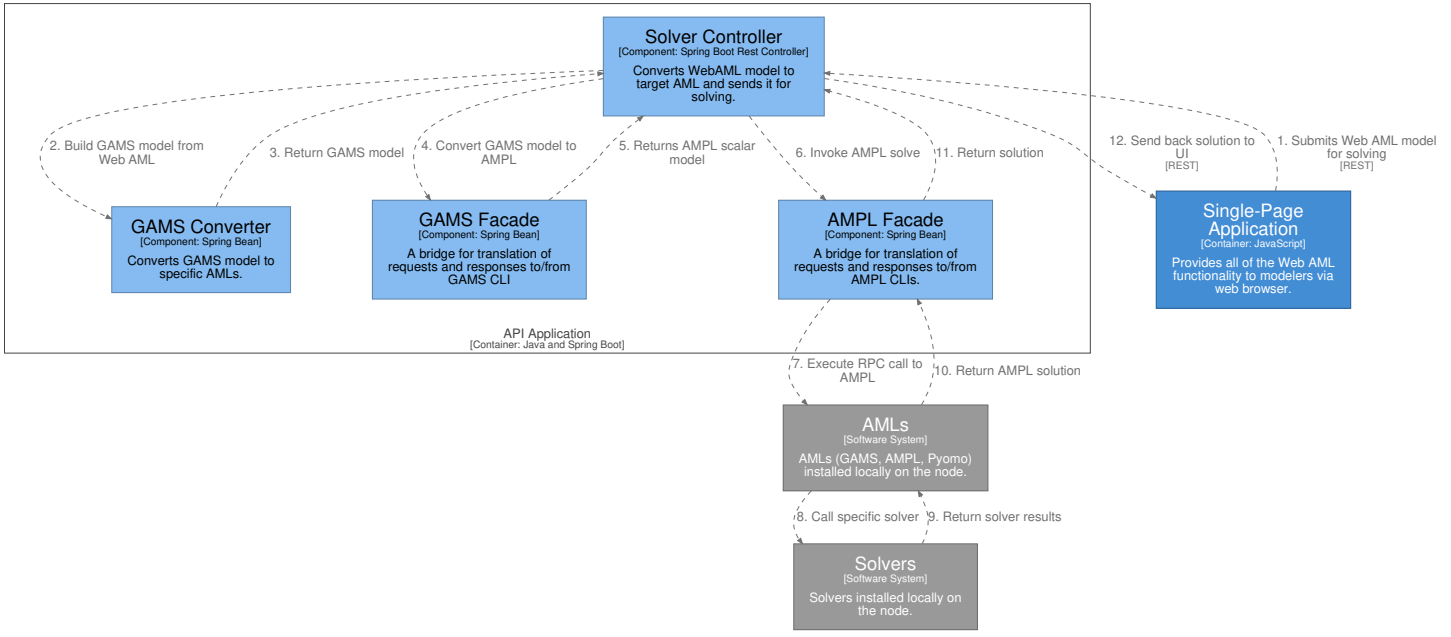

Figure 9: End-to-end flow to solve WebAML model using AMPL. This showcases the usage of GAMS Convert if a native WebAML to AMPL converter is not available.

When implementing the prototype, it was chosen to split the front-end and back-end parts of the prototype into:

- Client-side single page, React.js[22], the base application responsible for guiding the user while building a WebAML model;
- Back-end application implemented using Spring Boot[23] capable of parsing and converting WebAML language to other AMLs and communicating with underlying AMLs for solving the model.

This approach allows to leverage the strength of modern browser support for JSON and JSON Schema standards and offload the building of a WebAML model to the client-side, thus reducing the load on the back-end services.

The back-end also benefits from the fact that WebAML is a JSON-based language and can quickly validate incoming requests based on the standard JSON Schema while generating OpenAPI-based service contracts using it.

The prototype provides a simple yet straightforward and guided graphical user interface for constructing a WebAML model.

Since it was decided to use LaTeX for storing mathematical equations, an on-the-fly MathML-based visualizer of LaTeX equations was included in the prototype. It helps validate the mathematical expressions and guides not savvy LaTeX practitioners in the modeling process.

Examples of how the user interface looks are provided in Section 5.5 where the similarities and differences between the universal optimization system and AMLs are discussed.

5.4 Extending the prototype

Architectural decisions made in the design of the WebAML Optimization System make it the foundation for a universal mathematical optimization toolkit.

One possible future research direction could be extending the proposed WebAML language to simplify the model definition process. This can be

[22] React: A JavaScript library for building user interfaces; https://reactjs.org

[23] Spring Boot: a framework for building stand-alone, production-grade Spring-based applications; https://spring.io/projects/spring-boot

done by improving the user interface (e.g., adding textual guidance) and simplifying the syntax of the language (e.g., support for the implicit definition of sets).

Another research avenue could be the development of extensions to the universal optimization system. As an example, previous analysis revealed a need for parallel model generation and presolving capabilities.

Further research could be implementing automated solver algorithm selection in the WebAML Optimization System, which would significantly simplify the work for practitioners. Research in automatic algorithm selection is already ongoing and advanced [46].

Finally, testing in real-life situations and improving the existing prototype are needed to make the tool be adopted in mathematics classrooms and by enterprise users.

```
"sets": {
...
    "items": {
        ...
        "properties": {
            ...
            "valueType": {
                "type": "string",
                "title": "Data type of values in a set",
                // Add new boolean type to list below
                "enum": ["STRING", "NUMBER", "BOOLEAN"]
            },
            ...
        }
    }
}
```

Listing 9: WebAML JSON Schema extended with a new data type for the set component.

Practically extending the prototype with the required features could be done in multiple ways. Firstly, one can easily extend WebAML language based on JSON Schema by introducing new types for the already defined

essential components. Listing 9 shows how one could add a new data type called `boolean` to the `set` component.

Secondly, one can easily extend back-end services to support new AMLs by writing a converter from WebAML to the new AML, thus implementing the `WebAMLConverter` interface and implementing `AmlFacade` interface for integration with the underlying AML binaries. It does not require to do any changes in the controller or user interface. After fully implementing the `AmlFacade` interface provided in Listing 10, everything is registered automatically and works out of the box. There is even an option not to write a special WebAML converter, but to use the `GAMS CONVERT` tool as demonstrated in the `AmplConverter` class.

```
public interface AmlFacade {
    boolean isAmlAvailable();
    String getAmlName();
    List<String> getAvailableSolvers();
    AmlResult solveModel(String model, String solver);
    String convertModel(String model, String targetAml);
}
```

Listing 10: Structure of `AmlFacade` interface.

Finally, one can quickly introduce new features such as presolving or parallel solving by extending the existing `ModelController` class and adding additional features as an intermediate step between convert and solve operations.

5.5 Comparison with AMLs

To verify that in this research, it was succeeded to propose a universal web-based optimization system that does not require any prior knowledge of a specific AML syntax and provides a guided user interface for defining the model, the classical transportation problem by Dantzig, G. B. [15] was taken for a comparison once more.

In Section 1.3 a concrete instance of a transportation problem is described

highlighting what is needed to be modeled (Listing 1) and how it is modeled using `AMPL` and `JuMP` algebraic modeling languages (Listings 2 and 3).

In WebAML proposal, an approach contrary to the one used by AMLs for capturing optimization problems in a textual format is chosen. Instead, a graphical user interface is provided where the practitioner is guided through the problem definition process and given clear guidance on which information has to be supplied for a specific construct of the model.

A basic example of such an interface implemented in the prototype of a universal optimization system can be seen in Figure 10. Here it can be seen how the supply constraint and objective function of a given transportation model instance are captured using the user interface. The user is asked the basic information, such as the type of constraint, the indices used in the constraints, and the mathematical expression of the constraint itself.

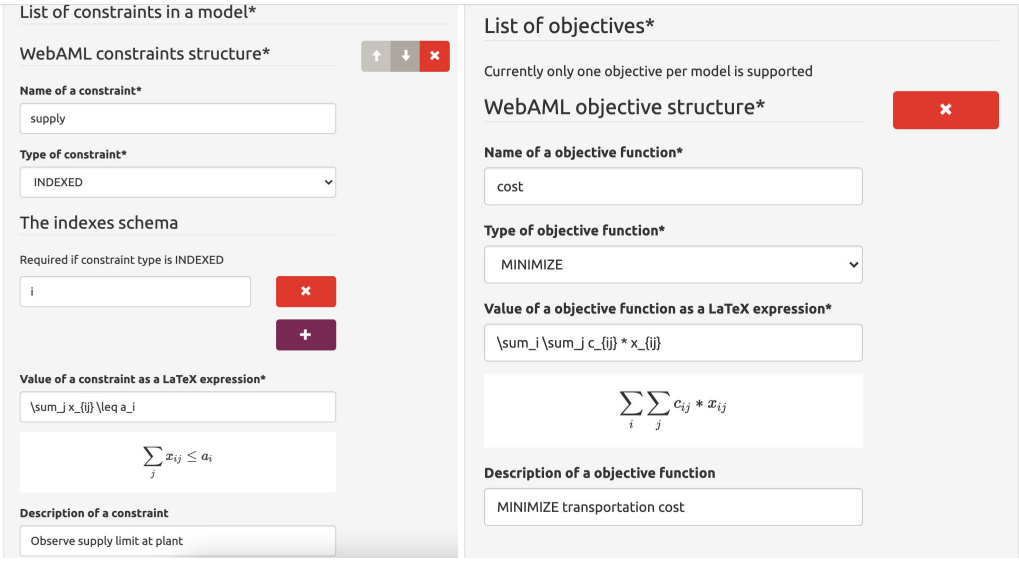

Figure 10: Screenshot of the user interface for WebAML modeling and optimization tool. Supply constraint and objective function of transportation problem are presented.

The constraints are entered using the LaTeX mathematical syntax displayed in a visual form to help the user and ensure that valid syntax is used. The objective function is entered similarly. In essence, each type of model component (sets, parameters, constraints, etc.) has its visual representation so

that the user experience can be custom-tailored to specific kinds of model components.

An example of how the solution to the problem and the solver output are displayed can be seen in Figure 11. This screen is used after the model has been defined. Here the user first has to choose which underlying AML to use for solving the model (`AMPL` in the example). Next, the user is presented with a list of solvers installed on the system compatible with a given AML. Lastly, after clicking *Solve* solution and the verbose output of AML and solver are provided to the user.

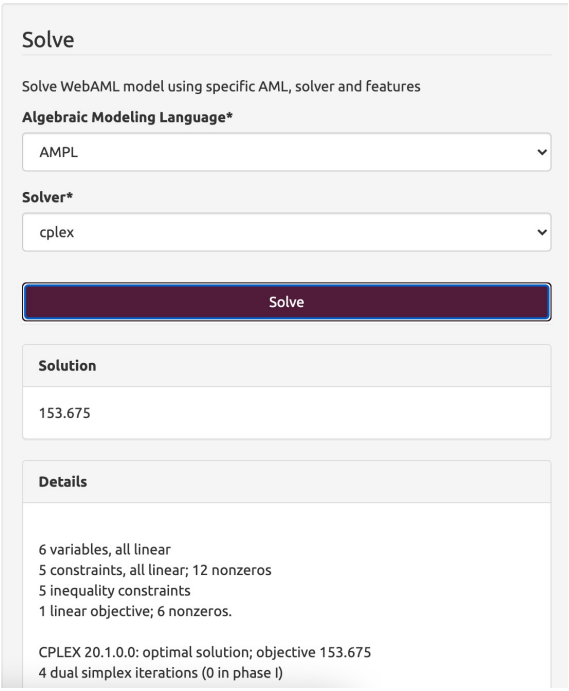

Figure 11: Screenshot of the user interface for WebAML modeling and mathematical optimization tool. Solution results of the transportation problem are presented.

It is important to note that this is only the prototype version of the user interface. If needed, it can be easily extended, thus making the process even smoother by providing guiding text and introducing questionnaire-like behavior and similar user experience improvements.

A full comparison of the characteristics exhibited by the proposed universal

optimization system and the AMLs are provided in Table 26.

Table 26: Comparison of characteristics in AMLs and universal modeling system (WebAML). Italic indicates features not implemented in the prototype.

	AMPL	**GAMS**	**JuMP**	**Pyomo**	**WebAML**
Graphical UI	No	No	No	No	Yes
Bespoke syntax	Yes	Yes	Yes	Yes	No
Compatibility	No	Yes	No	No	Yes
# of solvers	47	35	14	25	All
Presolving	Yes	No	No	No	*Yes*
Parallelism	No	No	No	No	*Yes*

A universal optimization system supporting all solvers means it is capable to utilize all solvers supported by the underlying AMLs.

Compatibility describes the capability of converting a model from one AML to another. `GAMS` can do this using `GAMS CONVERT` tool, while prototype is capable of both use `GAMS CONVERT`, but also constructing a model from WebAML language to target AML on its own.

Presolving and support for parallel scenario solving or parallel model generation are not implemented in the prototype. However, clear extension points are allowing other researchers to add those features.

5.6 Conclusions

This chapter set the requirements and described a proposal for the universal optimization system which consists of WebAML language and a prototype of the optimization system.

First, a formal JSON based language capable of capturing problem characteristics called WebAML was proposed. It was demonstrated how such language could be extended to support more features.

Next, an open-source and web-based prototype acting as an orchestrator that allows the AML modeler to build the model, convert it to a specific AML format, and send it for solving via local solvers or remote solving in the NEOS Server platform was presented.

Lastly, comparison between WebAML and other AMLs was done.
The following conclusions have been made:

1. The proposed formal language is capable of capturing problem characteristics in JSON format while the prototype allows to construct the model using WebAML language and solve it using the underlying AMLs. For now, WebAML language supports only basic features, however, being based on JSON Schema it is flexibly extendable.

2. Universal optimization system is not building any new solving capabilities, instead of relying and building on top of the best features provided by the most prominent AMLs. The tool also supports all solvers available in the underlying AMLs, thus providing a wide range of free and commercial solvers.

3. Universal optimization system does not require any specific algebraic language knowledge and allows for solving problems using different mathematical optimization solvers. Thus, it simplifies the process of algebraic modeling and mathematical optimization, making it available for individuals without detailed technical knowledge. This makes it appealing not only for enterprise users but also for teachers, lecturers, and students trying to understand the basics of mathematical optimization. Worldwide research on the usage of information and communication technology (ICT) to support, enhance, and optimize information delivery has shown that ICT can lead to improved student learning and better teaching methods (e.g., [61]).

4. Architectural decisions made in the design of the prototype make it the foundation for a universal mathematical optimization toolkit.

5. Since the tool can easily be installed on a server and accessed via a web interface, an institution can acquire a full academic or commercial license and allow easy access for every member to solve large-scale optimization problems.

GENERAL CONCLUSIONS

1. In this research, a concept of a universal optimization system that consists of a formalized algebraic modeling language (WebAML) and an open-source tool (WebAML Optimization System) for algebraic modeling and mathematical optimization was proposed.

   (a) Such system is aimed to provide a user-friendly environment simplifying mathematical modeling and optimization, but keeping the best features of the underlying AMLs.

   (b) System supports all solvers provided by underlying AMLs allowing practitioners to choose between the widest possible range of solvers. This allows practitioners to easily experiment with solving problems of different types and find the best solver to do it.

   (c) System is capable to convert a model from one AML to another, either by constructing a model from WebAML language to target AML using internal converters or by using `GAMS CONVERT` tool. In such a way allowing practitioners to easily move between different AMLs and use the one providing most features or being the best performing one.

   (d) The tool does not require any specific algebraic language knowledge and allows solving problems using different AMLs and optimization solvers. Thus, making it easier to learn and be taught in a mathematics classroom and still making it suitable for a usual practitioner once faced with a need for solving real-life mathematical optimization problems.

2. To prove the viability of such concept the prototype of the universal optimization system was developed, implementing the key features of the proposed concept and also providing clear extension points and ideas on how such a tool could be further developed.

   (a) The proposed WebAML language can be extended to simplify the model definition process. This can be done by improving the user

interface (e.g., adding textual guidance) and simplifying the syntax of the language (e.g., support for the implicit definition of sets).

(b) Prototype can be extended to support additional features as parallel model generation or presolving, or include more AMLs by writing converters from WebAML to the target AML. All this can be achieved by implementing already defined module interfaces and incorporating in into the prototype by configuration using dependency injection.

(c) Further research could also be made attempting to implement automated solver algorithm selection in the WebAML Optimization System, which would significantly simplify the work for practitioners.

3. While analyzing requirements for a universal optimization system an experimental analysis and multiples benchmarks of existing AMLs were conducted in order to evaluate different characteristics of AMLs: model size and verbosity, model instance creation time, presolving, and its impact on solving.

(a) Models implemented in `AMPL`, `GAMS`, and `JuMP` are the most compact ones, while the model written in `Pyomo` is more verbose. Comparing the number of language primitives required to create a model, `JuMP` and `AMPL` showed the best results. This could indicate that these modeling languages might have a more gentle learning curve, also allowing to conclude that in the context of the reviewed algebraic modeling languages, `JuMP` and `AMPL` enable practitioners to formulate an optimization problem most concisely.

(b) Model instance creation performance benchmark identified `AMPL` being a top performer, while `JuMP` and `Pyomo` performing the worst. There are no significant variations between different optimization problem types except for `JuMP`, where the model instance creation time tends to vary significantly while working with different types of problems.

(c) Performance difference between AMPL and other contenders increases when the models become larger. Comparing instance creation times of large models (models having more than 500 equations, 8 such models in the testing library) reveals 11 times the difference between AMPL and GAMS, 38 times the difference between AMPL and Pyomo, and close to 100 times the difference between AMPL and JuMP. The difference between GAMS and Pyomo stayed roughly the same - around 3.5 times.

(d) JuMP performance benchmark results confirm Dunning et al. statement that JuMP has a noticeable start-up cost of a few seconds even for the smallest instances. In this research, only the initialization of the JuMP package took around 7 seconds. A significant speed-up in multiple consecutive model instances creation was also observed. So potentially this could be mitigated when a family of models is solved multiple times within a single session, and compilation cost is only paid for the first time that an instance is solved.

(e) AMPL being the only AML supporting presolving out-of-the-box managed to simplify the models in 52.8% of the cases, out of which 5 times out of 7 it could determine that the problem solution is not feasible, thus not even requiring to call the solver. On average, once applied, the AMPL presolver managed to reduce the model size by removing 18.42% of constraints and 10.73% of variables. Overall, AMPL presolve positively impacted 26.43% of the cases iteration-wise and 47.86% time-wise. However, it produced a negative impact in 20.71% of cases iteration-wise and 23.57% time-wise. Still, the positive impact is greater than the negative one, especially in situations where the solver does not have problem presolving algorithms.

REFERENCES

[1] Kumar Abhishek, Sven Leyffer, and Jeff Linderoth. FilMINT: An outer approximation-based solver for convex mixed-integer nonlinear programs. *INFORMS Journal on computing*, 22(4):555–567, 2010. doi: 10.1287/ijoc.1090.0373.

[2] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 2019. doi: 10.1287/ijoc.2018.0857.

[3] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1): 1–36, March 2019. ISSN 1867-2949, 1867-2957. doi: 10.1007/s12532-018-0139-4. URL http://link.springer.com/10.1007/s12532-018-0139-4.

[4] Massimo Aria and Corrado Cuccurullo. Bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11 (4):959–975, 2017. ISSN 1751-1577. doi: 10.1016/j.joi.2017.08.007.

[5] Ateji. OptimJ, 2006. URL https://swmath.org/software/4917.

[6] Monya Baker. Reproducibility crisis. *Nature*, 533(26):353–66, 2016.

[7] Thomas Bartz-Beielstein, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, William La Cava, Manuel Lopez-Ibanez, et al. Benchmarking in optimization: Best practice and open issues. *arXiv preprint arXiv:2007.03488*, 2020.

[8] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format, December 2017. URL https://rfc-editor.org/rfc/rfc8259.txt.

[9] Michael R. Bussieck, Michael C. Ferris, and Alexander Meeraus. Grid-Enabled Optimization with GAMS. *INFORMS Journal on Computing*,

21(3):349–362, 2009. doi: 10.1287/ijoc.1090.0340. URL https://doi.org/10.1287/ijoc.1090.0340.

[10] Michael R. Bussieck, Michael C. Ferris, and Timo Lohmann. GUSS: Solving Collections of Data Related Models Within GAMS. In Josef Kallrath, editor, *Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems*, pages 35–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-23592-4_3. URL https://doi.org/10.1007/978-3-642-23592-4_3.

[11] Bussieck, Michael. Solving large-scale gams models on hpc platforms, 2019. URL https://www.gams.com/archives/presentations/GAMS_HPC_INFORMS2019.pdf.

[12] Bussieck, Michael and Fiand, Fred. High Performance Computing with GAMS, October 2017. URL https://www.gams.com/archives/presentations/informs2017_HPC_with_GAMS.pdf.

[13] Marco Colombo, Andreas Grothey, Jonathan Hogg, Kristian Woodsend, and Jacek Gondzio. A structure-conveying modelling language for mathematical and stochastic programming. *Mathematical Programming Computation*, 1(4):223–247, December 2009. ISSN 1867-2957. doi: 10.1007/s12532-009-0008-2. URL https://doi.org/10.1007/s12532-009-0008-2.

[14] Ovidiu Cosma, Petrică C. Pop, and Daniela Dănciulescu. A Parallel Algorithm for Solving a Two-Stage Fixed-Charge Transportation Problem. *Informatica*, 31(4):681–706, 2020. doi: 10.15388/20-INFOR432.

[15] George B Dantzig. The Classical Transportation Problem. In *Linear Programming and Extensions*, pages 299–315. Princeton University Press, 1963. ISBN 978-1-4008-8417-9. doi: 10.1515/9781400884179-015.

[16] RE Day and H Paul Williams. Magic: The design and use of an interactive modelling language for mathematical programming. *IMA Journal of Management Mathematics*, 1(1):53–65, 1986.

[17] Arne Drud, Stavros Zenios, and John Mulvey. *Balancing large social accounting matrices with nonlinear network programming*. The World Bank, 1986.

[18] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, 59(2):295–320, 2017. doi: 10.1137/15m1020575. URL https://doi.org/10.1137/15m1020575.

[19] Edinburgh Research Group in Optimization. SML: Structured Modelling Language, 2019. URL https://www.maths.ed.ac.uk/ERGO/sml.

[20] Daniele Fanelli. Is science really facing a reproducibility crisis, and do we need it to? *Proceedings of the National Academy of Sciences*, 115 (11):2628–2631, 2018.

[21] Pascual Fernández, Algirdas Lančinskas, Blas Pelegrín, and Julius Žilinskas. A Discrete Competitive Facility Location Model with Minimal Market Share Constraints and Equity-Based Ties Breaking Rule. *Informatica*, 31(2):205–224, 2020. doi: 10.15388/20-INFOR410.

[22] Michael C. Ferris and Todd S. Munson. Complementarity problems in GAMS and the PATH solver. *Journal of Economic Dynamics and Control*, 24(2):165–188, February 2000. ISSN 01651889. doi: 10.1016/S0165-1889(98)00092-X. URL https://linkinghub.elsevier.com/retrieve/pii/S016518899800092X.

[23] Christodoulos A Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[24] Robert Fourer. *AMPL : a modeling language for mathematical programming*. Thomson/Brooks/Cole, Pacific Grove, CA, 2003.

[25] Robert Fourer. Algebraic Modeling Languages for Optimization. In *Encyclopedia of Operations Research and Management Science*, pages 43–51. Springer, US, 2013. doi: 10.1007/978-1-4419-1153-7.

[26] Robert Fourer. Linear Programming: Software Survey. *OR/MS Today*, 44(3), June 2017. URL https://www.informs.org/ORMS-

Today/Public-Articles/June-Volume-44-Number-3/Linear-Programming-Software-Survey.

[27] Emmanuel Fragniere and Jacek Gondzio. Optimization Modeling Languages. *Handbook of Applied Optimization*, pages 993–1007, 2002.

[28] GAMS Development Corporation. GAMS Convert, 2019. URL https://www.gams.com/latest/docs/S_CONVERT.html.

[29] GAMS Development Corporation. GAMS Model Library, 2019. URL https://www.gams.com/latest/gamslib_ml/libhtml/index.html.

[30] GAMS Development Corporation. The Gather-Update-Solve-Scatter, 2020. URL https://www.gams.com/latest/docs/S_GUSS.html.

[31] GAMS Development Corporation. The Grid and Multi-Threading Solve Facility, 2020. URL https://www.gams.com/latest/docs/S_GUSS.html.

[32] David M Gay. Writing .nl files. *Optimization and Uncertainty Estimation*, 2005.

[33] Andreas Grothey and Feng Qiang. PSMG: A parallel problem generator for structure conveying modelling language for mathematical programming. *presentation at IC-COPT 2013*, 2009.

[34] Chris Groër, Bruce Golden, and Edward Wasil. A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing*, 23(2): 315–330, 2011. doi: 10.1287/ijoc.1100.0402.

[35] Francisco José Orts Gómez, Gloria Ortega López, Ernestas Filatovas, Olga Kurasova, and Gracia Ester Martın Garzón. Hyperspectral Image Classification Using Isomap with SMACOF. *Informatica*, 30(2): 349–365, 2019. doi: 10.15388/Informatica.2019.209.

[36] Gabriel Hackebeil. Parallel formulation of constraints and memory usage, April 2016. URL https://groups.google.com/d/msg/pyomo-forum/2ayvSKw-PKw/e4wuIdjpDQAJ.

[37] William E. Hart, Jean-Paul Watson, and David L. Woodruff. Pyomo: Modeling and Solving Mathematical Programs in Python. *Mathematical Programming Computation*, 3(3):219–260, September 2011. ISSN 1867-2949, 1867-2957. doi: 10.1007/s12532-011-0026-8.

[38] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, US, second edition, 2017.

[39] Eligius MT Hendrix, G Boglárka, et al. *Introduction to nonlinear and global optimization*, volume 37. Springer, 2010.

[40] Janine Huisman and Jeroen Smits. Duration and quality of the peer review process: the author's perspective. *Scientometrics*, 113(1): 633–650, 2017.

[41] Tony Hürlimann. *Mathematical Modeling and Optimization*, volume 31 of *Applied Optimization*. Springer US, Boston, MA, 1999. ISBN 978-1-4419-4814-4 978-1-4757-5793-4. doi: 10.1007/978-1-4757-5793-4.

[42] Vaidas Jusevičius and Remigijus Paulavičius. vaidasj/alg-mod-rev: Algebraic Modeling Language Benchmark, October 2020. URL https://zenodo.org/record/4106728.

[43] Vaidas Jusevičius and Remigijus Paulavičius. vaidasj/webaml: WebAML Tool for Algebraic Modeling Languages, September 2021. URL https://zenodo.org/record/5500339.

[44] Josef Kallrath, Panos M. Pardalos, and Donald W. Hearn, editors. *Modeling Languages in Mathematical Optimization*, volume 88 of *Applied Optimization*. Springer US, Boston, MA, 2004. ISBN 978-1-4613-7945-4 978-1-4613-0215-5. doi: 10.1007/978-1-4613-0215-5.

[45] G. Kannan, P. Sasikumar, and K. Devika. A genetic algorithm approach for solving a closed loop supply chain model: A case of battery recycling. *Applied Mathematical Modelling*, 34(3):655–670, March 2010. ISSN

0307904X. doi: 10.1016/j.apm.2009.06.021. URL https://linkinghub.elsevier.com/retrieve/pii/S0307904X0900170X.

[46] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1):3–45, 3 2019. ISSN 1063-6560, 1530-9304. doi: 10.1162/evco_a_00242. URL https://direct.mit.edu/evco/article/27/1/3-45/1083.

[47] Hossein Khodaei, Mahdi Hajiali, Ayda Darvishan, Mohammad Sepehr, and Noradin Ghadimi. Fuzzy-based heat and power hub models for cost-emission operation of an industrial consumer using compromise programming. *Applied Thermal Engineering*, 137:395–405, June 2018. ISSN 13594311. doi: 10.1016/j.applthermaleng.2018.04.008. URL https://linkinghub.elsevier.com/retrieve/pii/S1359431117379164.

[48] Sean Laraway, Susan Snycerski, Sean Pradhan, and Bradley E Huitema. An overview of scientific reproducibility: Consideration of relevant issues for behavior science/analysis. *Perspectives on Behavior Science*, 42(1):33–57, 2019.

[49] Kun-Young Lee, Ji-Soo Lim, and Sung-Seok Ko. Endosymbiotic Evolutionary Algorithm for an Integrated Model of the Vehicle Routing and Truck Scheduling Problem with a Cross-Docking System. *Informatica*, 30(3):481–502, 2019. doi: 10.15388/Informatica.2019.215.

[50] LINDO Systems, Inc. OptimJ, 2022. URL https://www.lindo.com/index.php/products/lingo-and-optimization-modeling.

[51] Marco Locatelli and Fabio Schoen. *Global optimization: theory, algorithms, and applications*. SIAM, 2013.

[52] Manuel López-Ibáñez, Juergen Branke, and Luís Paquete. Reproducibility in evolutionary computation. *ACM Transactions on Evolutionary Learning and Optimization*, 1(4):1–21, 2021.

[53] Miles Lubin. JuMP.jl, February 2017. URL https://github.com/JuliaOpt/JuMP.jl/issues/958#issuecomment-277267129.

[54] Miles Lubin and Iain Dunning. Computing in Operations Research Using Julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015. doi: 10.1287/ijoc.2014.0623.

[55] Bruce A McCarl, Alex Meeraus, Paul van der Eijk, Michael Bussieck, Steven Dirkse, and Franz Nelissen. *McCarl Expanded GAMS user guide*. Citeseer, US, 2016.

[56] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

[57] Robert R. Miner, David Carlisle, and Patrick D. F. Ion. Mathematical Markup Language (MathML) Version 3.0 2nd Edition. W3C Recommendation, W3C, April 2014.

[58] Hugo Morais, Péter Kádár, Pedro Faria, Zita A. Vale, and H.M. Khodr. Optimal scheduling of a renewable micro-grid in an isolated load area using mixed-integer linear programming. *Renewable Energy*, 35(1): 151–156, January 2010. ISSN 09601481. doi: 10.1016/j.renene.2009.02.031. URL https://linkinghub.elsevier.com/retrieve/pii/S0960148109001001.

[59] Artur Olszak and Andrzej Karbowski. Parampl: A Simple Tool for Parallel and Distributed Execution of AMPL Programs. *IEEE Access*, 6: 49282–49291, September 2018. doi: 10.1109/ACCESS.2018.2868222.

[60] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and David Moher. The PRISMA 2020 statement: an updated guideline for reporting systematic

reviews. *BMJ*, 372:n71, 2021. doi: 10.1136/bmj.n71. URL https://www.bmj.com/content/372/bmj.n71.

[61] Valeria Pandolfini. Exploring the Impact of ICTs in Education: Controversies and Challenges. *Italian Journal of Sociology of Education*, 8(06/2016):28–53, 2016. ISSN 2035-4983. doi: 10.14658/pupj-ijse-2016-2-3. URL https://doi.org/10.14658/pupj-ijse-2016-2-3.

[62] R Paulavičius, J Gao, P-M Kleniati, and C. S Adjiman. BASBL: Branch-And-Sandwich BiLevel solver: Implementation and computational study with the BASBLib test set. *Computers & Chemical Engineering*, 132: 106609, 2020. doi: 10.1016/j.compchemeng.2019.106609.

[63] Remigijus Paulavičius, Ya. D. Sergeyev, Dmitri E. Kvasov, and Julius Žilinskas. Globally-biased DISIMPL algorithm for expensive global optimization. *Journal of Global Optimization*, 59(2-3):545–567, 2014. doi: 10.1007/s10898-014-0180-4.

[64] Remigijus Paulavičius, Yaroslav D. Sergeyev, Dmitri E. Kvasov, and Julius Žilinskas. Globally-biased BIRECT algorithm with local accelerators for expensive global optimization. *Expert Systems with Applications*, 144:113052, 2020. doi: 10.1016/j.eswa.2019.113052.

[65] Remigijus Paulavičius and Julius Žilinskas. *Simplicial Global Optimization*. SpringerBriefs in Optimization. Springer, New York, 2014. ISBN 978-1-4614-9092-0. doi: 10.1007/978-1-4614-9093-7.

[66] Yash Puranik and Nikolaos V. Sahinidis. Domain reduction techniques for global NLP and MINLP optimization. *Constraints*, 22(3):338–376, 2017. doi: 10.1007/s10601-016-9267-5.

[67] J VANDERBEI ROBERT. *Linear programming: Foundations and extensions*. Springer, 2021.

[68] Edward Rothberg. How A Mathematical Optimization Model Can Help Your Business Deal With Disruption, 2020. URL https://forbes.com/sites/forbestechcouncil/2020/08/24/how-a-

mathematical-optimization-model-can-help-your-business-deal-with-disruption.

[69] Mohammadhossein Saeedi, Mahdi Moradi, Meysam Hosseini, Armin Emamifar, and Noradin Ghadimi. Robust optimization based optimal chiller loading under cooling demand uncertainty. *Applied Thermal Engineering*, 148:1081–1091, February 2019. ISSN 13594311. doi: 10.1016/j.applthermaleng.2018.11.122. URL https://linkinghub.elsevier.com/retrieve/pii/S1359431118353547.

[70] Linas Stripinis, Remigijus Paulavičius, and Julius Žilinskas. Penalty functions and two-step selection procedure based DIRECT-type algorithm for constrained global optimization. *Structural and Multidisciplinary Optimization*, 59(6):2155–2175, 2019. doi: 10.1007/s00158-018-2181-2.

[71] Linas Stripinis, Julius Žilinskas, Leocadio G Casado, and Remigijus Paulavičius. On MATLAB experience in accelerating DIRECT-GLce algorithm for constrained global optimization through dynamic data structures and parallelization. *Applied Mathematics and Computation*, 390:125596, 2021. doi: 10.1016/j.amc.2020.125596.

[72] StructJuMP. StructJuMP, 2020. URL https://github.com/StructJuMP/StructJuMP.jl.

[73] Tomlab. OptimJ, 2020. URL https://tomopt.com/.

[74] Charalampos P. Triantafyllidis and Lazaros G. Papageorgiou. An integrated platform for intuitive mathematical programming modeling using LaTeX. *PeerJ Computer Science*, 4:e161, September 2018. ISSN 2376-5992. doi: 10.7717/peerj-cs.161. URL https://peerj.com/articles/cs-161.

[75] Zsolt Ugray, Leon Lasdon, John Plummer, Fred Glover, James Kelly, and Rafael Martí. Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. *INFORMS Journal on Computing*, 19(3):328–340, August 2007. ISSN 1091-9856, 1526-5528. doi: 10.

1287/ijoc.1060.0175. URL http://pubsonline.informs.org/doi/10.1287/ijoc.1060.0175.

[76] Nees Jan van Eck and Ludo Waltman. Software survey: VOSviewer, a computer program for bibliometric mapping. *Scientometrics*, 84(2): 523–538, 8 2010. ISSN 1588-2861. doi: 10.1007/s11192-009-0146-3.

[77] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1): 67–82, 4 1997. ISSN 1089778X. doi: 10.1109/4235.585893. URL http://ieeexplore.ieee.org/document/585893/.

[78] AJ Wood and BF Wollenberg. *Power generation, operation, and control*. John Wiley and Sons Inc., New York, NY, 1984.

[79] Austin Wright, Henry Andrews, Ben Hutton, and Greg Dennis. JSON Schema: A Media Type for Describing JSON Documents. Internet-Draft draft-bhutton-json-schema-00, Internet Engineering Task Force, December 2020. URL https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-00.

[80] Kaan Yetilmezsoy, Sevgi Demirel, and Robert J. Vanderbei. Response surface modeling of Pb(II) removal from aqueous solution by Pistacia vera L.: Box–Behnken experimental design. *Journal of Hazardous Materials*, 171(1-3):551–562, November 2009. ISSN 03043894. doi: 10.1016/j.jhazmat.2009.06.035. URL https://linkinghub.elsevier.com/retrieve/pii/S0304389409009480.

[81] H.H. Zeineldin, E.F. El-Saadany, and M.M.A. Salama. Optimal coordination of overcurrent relays using a modified particle swarm optimization. *Electric Power Systems Research*, 76(11):988–995, July 2006. ISSN 03787796. doi: 10.1016/j.epsr.2005.12.001. URL https://linkinghub.elsevier.com/retrieve/pii/S0378779605002701.

APPENDIX A

Models of the transportation problem

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

Parameter
   a(i) 'capacity of plant i in cases'
        / seattle    350
          san-diego  600 /

   b(j) 'demand at market j in cases'
        / new-york   325
          chicago    300
          topeka     275 /;

Table d(i,j) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

Variable
   x(i,j) 'shipment quantities in cases'
   z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
   cost      'define objective function'
   supply(i) 'observe supply limit at plant i'
   demand(j) 'satisfy demand at market j';

cost..      z =e= sum((i,j), c(i,j)*x(i,j));
supply(i).. sum(j, x(i,j)) =l= a(i);
demand(j).. sum(i, x(i,j)) =g= b(j);
Model transport / all /;
solve transport using lp minimizing z;
```

Listing 11: Transportation problem defined in GAMS format

```
from pyomo.environ import *

model = ConcreteModel()

model.i = Set(initialize=['seattle','san-diego'])
model.j = Set(initialize=['new-york','chicago', 'topeka'])

model.a = Param(model.i, initialize={'seattle':350,'san-diego':600})
model.b = Param(model.j, initialize={'new-york':325,'chicago':300,
    'topeka':275})

dtab = {
    ('seattle',  'new-york') : 2.5,
    ('seattle',  'chicago')  : 1.7,
    ('seattle',  'topeka')   : 1.8,
    ('san-diego','new-york') : 2.5,
    ('san-diego','chicago')  : 1.8,
    ('san-diego','topeka')   : 1.4,
    }
model.d = Param(model.i, model.j, initialize=dtab)
model.f = Param(initialize=90)
def c_init(model, i, j):
  return model.f * model.d[i,j] / 1000
model.c = Param(model.i, model.j, initialize=c_init)

model.x = Var(model.i, model.j, bounds=(0.0,None))

def supply_rule(model, i):
  return sum(model.x[i,j] for j in model.j) <= model.a[i]
model.supply = Constraint(model.i, rule=supply_rule)
def demand_rule(model, j):
  return sum(model.x[i,j] for i in model.i) >= model.b[j]
model.demand = Constraint(model.j, rule=demand_rule)

def objective_rule(model):
  return sum(model.c[i,j]*model.x[i,j] for i in model.i for j in model.j)
model.objective = Objective(rule=objective_rule, sense=minimize)
```

Listing 12: Transportation problem defined in Pyomo format

APPENDIX B

Component diagram of the prototype

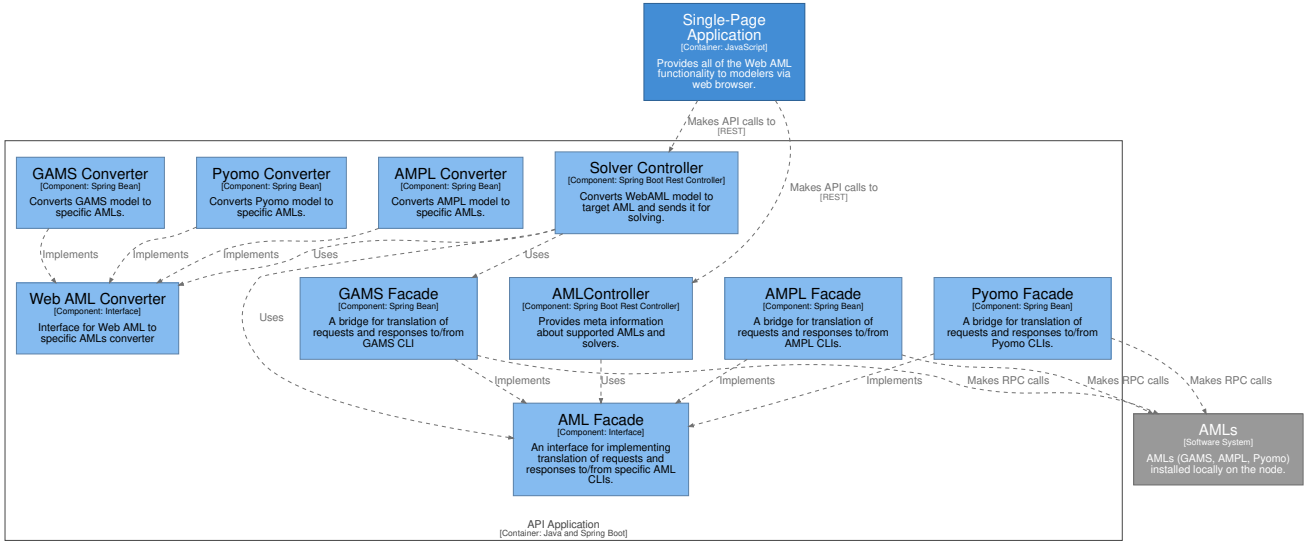

Figure 12: Components of the universal optimization system in C4 architectural model notation. WebAML Converter and AML Facade interfaces support additional new AMLs.

SUMMARY IN LITHUANIAN

Tyrimo sritis ir problemos aktualumas

Daugelis realaus pasaulio problemų yra reguliariai sprendžiamos naudojant šiuolaikinius optimizavimo įrankius [pvz., 1, 27, 34, 65, 62, 64]. Sprendžiant pateiktą problemą šiomis priemonėmis naudojamas matematinio modelio ir atitinkamo sprendimo algoritmo derinys [pvz., 14, 21, 35, 49, 65, 63, 62, 70, 71]. Todėl matematinių modelių formulavimo būdas yra labai svarbus optimizavimo poveikiui realiame gyvenime. Galimi realių problemų pavyzdžiai yra įmonių vykdoma gamyba ir prekių siuntimas, investicijų planavimas, makroekonomikos stabilizavimas, vandentiekio tinklai, naftos perdirbimo gamyklos, chemijos gamyklos, tarptautinė prekyba aliuminiu ir variu bei daugelis kitų [29].

Matematinis modeliavimas – tai realių verslo problemų pavertimo matematinėmis formuluotėmis procesas, kurio teorinė ir skaitinė analizė gali suteikti įžvalgų, atsakymų ir gairių, naudingų taikymui praktikoje [44], netgi ir analizuojant dabartinę COVID-19 pandemiją [68]. Algebrinio modeliavimo kalbos (AMK) yra deklaratyvios optimizavimo modeliavimo kalbos, kurios užpildo atotrūkį tarp modelio formulavimo ir uždavinio sprendinio radimo technikos [27]. Jos leidžia suformuluoti matematinį modelį kaip žmogaus skaitomą lygčių rinkinį, tačiau autoriui nereikia nurodyti, kaip aprašytas modelis turi būti išspręstas ar koks konkretus sprendėjas turi būti panaudotas.

AMK aprašyti modeliai yra žinomi dėl didelio panašumo į matematinę formuluotę. Šis aspektas išskiria AMK iš kitų modeliavimo kalbų tipų, tokių kaip objektiškai orientuotų (pvz., `OptimJ`), sprendėjui specifinių (pvz., `LINGO`) ar bendrosios paskirties (pvz., `TOMLAB`) modeliavimo kalbų. Toks algebrinio projektavimo metodas leidžia specialistams, neturintiems specifinių programavimo ar modeliavimo žinių, efektyviai aprašyti sprendžiamas problemas. Taip pat svarbu pažymėti, kad AMK yra atsakinga už problemos modelio egzemplioriaus, kurį gali išspręsti sprendimo algoritmas, sukūrimą [44]. Kadangi daugelis AMK yra neatskiriama konkrečios modeliavimo sistemos dalis, būtina modeliavimo kalbos

atsakomybę atskirti nuo visos sistemos. Apibendrinant, AMK yra sudėtingi programinės įrangos paketai, kurie palaiko esminį ryšį tarp optimizavimo modelio matematinės koncepcijos ir sudėtingų algoritminių veiksmų, apskaičiuojančių optimalius sprendimus. Paprastai AMK programinė įranga automatiškai nuskaito modelį ir duomenis, sugeneruoja egzempliorių ir perduoda jį sprendėjui jam suprantama forma [25].

Nuo 1970-ųjų buvo sukurta nemažai algebrinių modeliavimo kalbų (pvz., `GAMS` [55], `AMPL` [24]) ir vis dar yra aktyviai kuriamos naujos algebrinės modeliavimo kalbos. Pastaruoju metu pristatytos kelios naujos atvirojo kodo pagrindu sukurtos algebrinio modeliavimo kalbos (pvz., `Pyomo` [37, 38], `JuMP` [18, 54]), kurios sudaro konkurenciją tradicinėms komercinėms AMK. Todėl norint ištirti dabartinį algebrinių modeliavimo kalbų išsivystymo lygį reikia apžvelgti ir palyginti tradicines ir atsirandančias naujas AMK.

Iki šiol didžioji dalis AMK palyginimų buvo atlikti remiantis klausimynais, pateiktais AMK kūrėjams [26]. Tačiau autorius mano, jog trūksta išsamios teorinės ir eksperimentinės analizės, susijusios su ryškiausių AMK (`AMPL`, `GAMS`, `JuMP` ir `Pyomo`) ir jas palaikančių modeliavimo sistemų charakteristikomis.

Taip pat autorius mano, kad tikslinga tęsti tyrimus toliau šalinant pagrindines esamų AMK ir jas palaikančių optimizavimo sistemų spragas. Tai leistų nustatyti reikalavimus universalesnių optimizavimo sistemų koncepcijai, apjungiančiai geriausias esamų AMK savybes. Darbas šia kryptimi jau buvo pradėtas teikiant pasiūlymus naudoti LaTeX kaip AMK įrankio pagrindą, kurį pateikė Triantafyllidis ir Papageorgiou [74], arba CasADi [3], siūlantis atvirojo kodo įrankį netiesiniam optimizavimui ir algoritminiam diferencijavimui. Tačiau, autoriaus nuomone, galima pasiūlyti kitą – labiau praplečiamą ir patogesnę vartotojui alternatyvą, kuri būtų naudinga ne tik matematinį išsilavinimą turintiems specialistams, bet ir tiems, kurie tik pradeda mokytis matematinio optimizavimo.

Tyrimo objektas

Šios disertacijos tyrimo objektas yra programų sistemos, įgalinančios algebrinėmis modeliavimo kalbomis spręsti matematinio optimizavimo

uždavinius. Darbe nagrinėjama tiek komercinė, tiek ir atvirojo kodo algebrinio modeliavimo ir optimizavimo programinė įranga.

Darbo tikslai ir uždaviniai

Disertacijos darbo tikslas – pasiūlyti universalios optimizavimo sistemos koncepciją, apimančią algebrinio modeliavimo kalbą ir modeliavimo sistemą, jungiančią geriausias populiariausių AMK charakteristikas.

Siekiant įgyvendinti iškeltą tikslą numatyti šie uždaviniai:

1. Identifikuoti pagrindines šiuolaikinių algebrinio modeliavimo kalbų ypatybes ir parinkti žinomiausias šiuo metu rinkoje naudojamas ir kriterijus atitinkančias algebrinio modeliavimo kalbas.
2. Teoriškai apžvelgti pasirinktas algebrinio modeliavimo kalbas ir sistemas, palyginti jų skirtumus ir nustatyti kiekvienos trūkumus.
3. Sukurti optimizavimo uždavinių biblioteką, kuria remiantis būtų galima atlikti pasirinktų AMK našumo testus.
4. Atlikti eksperimentinę pasirinktų AMK analizę naudojantis našumo testų rezultatais.
5. Užtikrinti eksperimentinės analizės rezultatų atkuriamumą.
6. Remiantis teorine ir eksperimentine analize nustatyti ir aprašyti galimas esamų algebrinio modeliavimo sistemų tobulinimo gaires.
7. Pasiūlyti universalios optimizavimo sistemos koncepciją, apimančią bendrąją algebrinio modeliavimo kalbą ir ją palaikančią atvirojo kodo sistemą.
8. Sukurti siūlomos universalios optimizavimo sistemos prototipą, įrodantį jos įgyvendinamumą ir tinkamumą tolesniam visapusiškos universalios optimizavimo sistemos kūrimui.

Tyrimo metodai

Algebrinio modeliavimo ir matematinio optimizavimo sritims analizuoti taikyti informacijos paieškos, organizavimo, analizės, lyginamosios analizės, apibendrinimo metodai. Eksperimentinio tyrimo interpretavimui, siekiant įvertinti algebrinio modeliavimo kalbų efektyvumą, taikyta statistinė analizė.

Mokslinis darbo naujumas

Pagrindiniai mokslinio darbo naujumai yra šie:

1. Buvo atlikta išsami lyginamoji svarbiausių algebrinio modeliavimo kalbų bei jas palaikančių modeliavimo sistemų (`AMPL`, `GAMS`, `JuMP` ir `Pyomo`) panašumų ir skirtumų analizė. Iki šiol AMK palyginimai buvo riboti pagal vertinamų charakteristikų kiekį arba pagal tai, kaip jie buvo atlikti (pvz., sutelkiant dėmesį į kūrėjų atsakymus klausimynų pagrindu). Čia vienoje vietoje buvo įvertinta ne tik atitiktis pagrindiniams šiuolaikinės AMK reikalavimams, bet ir palygintos jų pagrindinės ir papildomos funkcijos, patogumas, perkeliamumas, kainodara.

2. Remiantis eksperimentiniu modelio egzempliorių kūrimo laiko tyrimu buvo sukurta atvira testavimo ir praktinio optimizavimo uždavinių (algebrinių modelių) biblioteka. Modeliai pateikiami keliais skirtingais formatais, kuriuos palaiko šiuolaikinių optimizavimo uždavinių sprendėjai. Esminis šios bibliotekos išskirtinumas yra net ne jos dydis ar modelių formatų įvairovė, o atvirumas, decentralizacija ir įrankių palaikymas. Atvirojo kodo ir *GitHub* pagrindu sukurta saugykla suteikia galimybę kiekvienam prisidėti prie šios bibliotekos augimo, o autorių pateikti aprašai ir įrankiai leidžia lengvai ją vystyti ir plėsti.
   Jusevičius, V.; Paulavičius, R. Vaidasj/Alg-Mod-Rev: Algebraic Modeling Language Benchmark, 2020. 10.5281/ZENODO.4106728; URL: https://github.com/vaidasj/alg-mod-rev

3. Iš operacijų tyrimo prizmės įvertintas modelio egzemplioriaus kūrimo efektyvumas ir modelio supaprastinimas. Eksperimentinis modelio

egzemplioriaus sukūrimo laiko tyrimas buvo atliktas su modelių biblioteka, kurią sudaro beveik trys šimtai modelių, apimančių skirtingus problemų tipus ir dydžius. Tai iki šiol didžiausias modelio egzemplioriaus sukūrimo laiko tyrimas. Jame akcentuojami reikšmingi AMK efektyvumo skirtumai ir identifikuojamos neatitiktys su AMK kūrėjų atliktais tyrimais. Kitu eksperimentiniu tyrimu siekta įverti AMPL išankstinio sprendėjo naudą modeliui supaprastinti. Nustatyta, kad teigiamas išankstinio sprendimo poveikis visada yra reikšmingesnis nei neigiamas.
Jusevičius, V.; Paulavičius, R. Vaidasj/Alg-Mod-Rev: Algebraic Modeling Language Benchmark, 2020. 10.5281/ZENODO.4106728; URL: https://github.com/vaidasj/alg-mod-rev

4. Remiantis lyginamąja ir eksperimentine analize buvo nustatyti ryškiausių AMK skirtumai ir trūkumai. Atsižvelgiant į gautus rezultatus pasiūlyta atvirojo kodo universalios optimizavimo sistemos koncepcija. Ji sujungia geriausias esamų algebrinio modeliavimo kalbų charakteristikas, taip pat suteikia intuityvų ir patogų optimizavimo problemų formulavimo (t. y. modelio apibrėžimo) procesą. Buvo apibrėžti du pagrindiniai koncepcijos elementai: WebAML kalba, skirta aprašyti problemos semantiką, ir optimizavimo sistema, veikianti kaip tarpininkas tarp WebAML kalbos ir kitų AMK.

5. Sukurtas analogų pasaulyje neturinčios universalios optimizavimo sistemos prototipas. Norint naudotis prototipu nereikia jokių specifinių algebrinės kalbos žinių. Taip pat prototipas įgalina spręsti problemas naudojant skirtingus matematinius optimizavimo sprendėjus. Jį gali naudoti tiek mokslininkai, tiek praktikai įvairiuose ūkio sektoriuose. Šis įrankis įgalina greitesnį ir efektyvesnį modelio kūrimą sprendimų priėmėjams.
Jusevičius, V.; Paulavičius, R. Vaidasj/WebAML: WebAML Tool for Algebraic Modeling Languages, 2021. 10.5281/ZENODO.5500339; URL: https://github.com/vaidasj/WebAML

Ginamieji teiginiai

1. Yra nemažai galingų modeliavimo aplinkų ir algebrinio modeliavimo kalbų, tačiau nė viena iš jų nesuteikia viso funkcijų rinkinio, reikalingo efektyviam ir intuityviam optimizavimo problemų modeliavimui.
2. Universali optimizavimo sistema, sujungianti geriausias kelių žinomų AMK charakteristikas, padeda pašalinti populiariausių AMK trūkumus.
3. Architektūriniai sprendimai, priimti kuriant WebAML prototipą, leidžia jį naudoti kaip universalaus optimizavimo įrankių rinkinio pagrindą. Įrankių rinkinys gali būti papildytas kitomis pažangiausiomis funkcijomis, kaip, pavyzdžiui, išankstinis sprendimas, paskirstytas sprendimas arba geriausio sprendėjo pasirinkimas pagal optimizavimo uždavinio tipą.

Darbo rezultatų aprobavimas

Pagrindiniai tyrimo rezultatai publikuoti dvejuose recenzuojamuose periodiniuose leidiniuose:

1. **Jusevičius, V.** ir Paulavičius, R. „Web-Based Tool for Algebraic Modeling and Mathematical Optimization“. Mathematics, 2021, 9 (21), 2751. DOI: 10.3390/math9212751.
2. **Jusevičius, V.**, Oberdieck, R. ir Paulavičius, R. „Experimental Analysis of Algebraic Modelling Languages for Mathematical Optimization“. Informatica, 2021, 32 (2), 283–304. DOI: 10.15388/21-INFOR447.

Pagrindiniai tyrimo rezultatai pristatyti dvejose tarptautinėse konferencijose:

1. **Jusevičius, V.** ir Paulavičius, R. „Experimental Analysis of Algebraic Modeling Languages For Social Behavior Modeling“, The International EURO mini Conference Modelling and Simulation of Social-Behavioural Phenomena in Creative Societies, rugsėjo 18–20, 2019. Vilnius, Lietuva.

2. **Jusevičius, V.** ir Paulavičius, R. „Web-based tool for algebraic modeling languages“, EURO 2021: 31st European Conference on Operational Research, liepos 11-13, 2021. Atėnai, Graikija.

Algebrinės modeliavimo kalbos

Šiame skyriuje analizuojamos algebrinio modeliavimo kalbos ir nustatomos pagrindinės AMK charakteristikos, kurios bus aptariamos ir analizuojamos kituose skyriuose.

Pirma, pabrėžiama algebrinio modeliavimo kalbų reikšmė. Svarbiausia, kad jos leidžia atskirti modelio formuluotę nuo įgyvendinimo detalių, išlaikant žymėjimą, artimą problemos matematinei formuluotei. AMK išskiria abstrakčius modelius ir konkrečius problemų atvejus, kai konkretus modelio egzempliorius generuojamas iš abstraktaus modelio naudojant duomenis. Tai leidžia sukurti daugkartinio naudojimo modelius, o matematinis optimizavimas yra priimtinesnis specialistams, neturintiems gilaus matematinio pagrindo.

Vėliau pateikiamas Dantzig klasikinės transportavimo problemos pavyzdys, aprašytas matematine sintakse. Tai yra tiesinio programavimo problema, kurios tikslas yra sumažinti transportavimo išlaidas, atsižvelgiant į paklausos ir pasiūlos ribojimus. Toliau konkretaus modelio egzemplioriaus pavyzdys aprašomas loginėmis sąvokomis ir paverčiamas į dviejų algebrinio modeliavimo kalbų `AMPL` ir `JuMP` žymėjimus. Pastebėta, kad gana stipriai skiriasi algebrinio modeliavimo kalbų sintaksė. Dviejuose pateiktuose pavyzdžiuose `AMPL`, atrodo, skirtas specialistams, turintiems matematinį išsilavinimą, o `JuMP` labiau tinka specialistams, kurie yra geriau susipažinę su įprastomis programavimo kalbomis.

Toliau apibūdinamos pagrindinės šiuolaikinių AMK charakteristikos, identifikuojamos ir pasirenkamos tolesniems teoriniams ir eksperimentiniams tyrimams svarbiausios algebrinės kalbos. Trys pagrindinės savybės yra tai, kad problemos aprašomos deklaratyviai; yra aiški atskirtis tarp problemos apibrėžimo ir sprendimo proceso bei egzistuoja aiškus problemos struktūros ir jos duomenų atskyrimas. Kitų funkcijų, pvz., automatinio diferencijavimo, poreikis taip pat akivaizdus, bet ne esminis. Remiantis nustatytais

reikalavimais, akademiniu pripažinimu ir naudojimo populiarumu buvo nustatytos keturios svarbiausios AMK: AMPL, GAMS, JuMP ir Pyomo.

Galiausiai, siekiant suprasti temos mastą, atsirandančias tendencijas ir raidą bėgant laikui, buvo atlikta pasirinktų AMK bibliometrinė analizė. Naudota adaptuota PRISMA (*Preferred Reporting Items for Systematic Reviews and Meta-Analysis*) [60] metodologijos versija, skirta literatūros paieškai mokslinėse duomenų bazėse. Buvo analizuojami straipsniai, publikuoti nuo 2000 iki 2021 metų. Iš viso išnagrinėta 2550 dokumentų, publikuotų 927 šaltiniuose (žurnaluose, knygose ir kt.). Nuo 2000 iki 2019 metų stebimas nuolat augantis publikacijų skaičius tyrimų srityje, tačiau pastaraisiais metais skaičiai mažėja, o tai kartu su autoriaus raktažodžių analize rodo, kad šios srities tyrimai konsoliduojami. Septyniuose iš dešimties dažniausiai cituojamų straipsnių GAMS buvo paminėtas kaip vienas iš raktinių žodžių. GAMS taip pat buvo tarp populiariausių raktinių žodžių, rodančių GAMS svarbą ir populiarumą. Atsižvelgiant į raktinių žodžių tendencijas matoma, kad artimiausioje ateityje tvarios energijos inžinerija bus pagrindinė matematinio optimizavimo ir algebrinio modeliavimo kalbų tyrimų kryptis.

Algebrinių modeliavimo kalbų lyginamoji analizė

Šiame skyriuje aptariami pagrindiniai keturių svarbiausių algebrinio modeliavimo kalbų (AMPL, GAMS, JuMP ir Pyomo) skirtumai.

Visų pirma, programinės įrangos paketai buvo lyginami atsižvelgiant į tokius aspektus kaip operacinės sistemos palaikymas, licencijavimas ir vartotojo sąsajos tipas. Visi jie palaiko tris pagrindines operacines sistemas (*Windows, Unix, Mac OS*), todėl naudojimas turėtų būti gana sklandus nepriklausomai nuo to, kokią operacinę sistemą ar techninę įrangą naudoja vartotojas. Be to, visi, išskyrus Pyomo, pateikia grafinę vartotojo sąsają tekstiniam modelio kodui rašyti ir kai kurioms standartinėms komandoms vykdyti. AMPL ir GAMS yra komercinės programinės įrangos, kurių minimali (bazinė) kaina yra nuo 3000 iki 4000 JAV dolerių, o pridėjus sprendėjų įkainį ji žymiai išauga. Bandomosios versijos yra ribotos naudojimo laiku arba modelio dydžiu. Net ir akademinis licencijavimas nėra nemokamas (jo kaina

– nuo 400 JAV dolerių). Pyomo ir JuMP yra atvirojo kodo ir platinamos be mokesčio, tačiau už komercinius sprendėjus reikia mokėti atskirai.

Toliau siekiant nustatyti, kaip nagrinėjamos AMK atitinka šiuolaikinėms AMK keliamus reikalavimus, buvo atlikta lyginamoji savybių analizė. Visose apžvelgtose AMK optimizavimo problemos pateikiamos deklaratyviai. Kadangi visos jos yra konkrečios modeliavimo sistemos dalis, šios sistemos kontekste problemos apibrėžimo ir sprendimo procesai aiškiai atskiriami. Problemos struktūra ir jos duomenys atskirti visose apžvelgtose kalbose. Visos nagrinėtos AMK taip pat leidžia modeliuoti problemas nepriklausomai nuo sprendėjų. Duomenų pateikimo modeliui būdai tarp AMK skiriasi. Nors visos jos palaiko įvestį iš failo, kai kurie sudėtingesni scenarijai, pvz., duomenų nuskaitymas iš reliacinių duomenų bazių, yra lengviau realizuojami JuMP arba Pyomo kalbose. Kalbant apie sprendėjų palaikymą, AMPL turi daugiausia palaikomų sprendėjų. Išankstinio sprendimo galimybės pasiekiamos tik AMPL. JuMP ir Pyomo turi programavimo sąsajas, leidžiančias realizuoti išankstinio sprendimo galimybes, bet šiuo metu tokių įrankių nėra sukurta. Naudojant Python arba Julia bibliotekas galima vizualizuoti JuMP ir Pyomo rezultatus, tačiau tą reikia realizuoti patiems, nes nei JuMP, nei Pyomo to standartiškai nepateikia. Galima daryti išvadą, kad visos nagrinėtos kalbos atitinka esmines šiuolaikinių AMK savybes.

Galiausiai buvo įvertintos daug žadančios lygiagretaus AMK veikimo savybės. Nustatyti trys pagrindiniai lygiagretinimo algebrinio modeliavimo panaudojimo atvejai kalbose, tačiau dėmesys sutelktas tik į du, kuriuos palaiko pačios AMK, o ne sprendėjai. Vienas panaudojimo atvejis yra problemų rinkinio optimizavimas, kai kiekviena problema yra struktūriškai ta pati, bet kai kurie arba visi duomenys, apibrėžiantys egzempliorių, yra atnaujinami. Antra – didelio masto problemos, kurioms reikia lygiagretaus apdorojimo ne tik problemos sprendimui, bet ir modelio kūrimo fazėje. Apžvelgti AMK turi ribotą palaikymą tokiam lygiagretinimui ir didžioji dalis to pasiekiama pasitelkus nestandartinių plėtinių arba esamų AMK savybių kompoziciją. Lygiagretus modelio kūrimas yra labai ankstyvoje brandos stadijoje ir kai kurie autoriai abejoja lygiagrečiojo modelio generavimo teikiama nauda. Tačiau atsižvelgiant į egzistuojančius našumo tyrimus galima teigti, kad tinkamai pritaikius lygiagretinimą našumas sprendžiant

realias matematines optimizavimo problemas gali padidėti.

Algebrinių modeliavimo kalbų eksperimentinė analizė

Šiame skyriuje pateikiamas testavimo uždavinių rinkinys, skirtas praktinių optimizavimo problemų sprendimo našumo analizei atlikti, ir pristatomi pasirinktų AMK eksperimentai ir jų išvados.

Pirmiausia pateikiamos autorių *GitHub* puslapyje publikuotos bibliotekos modelių charakteristikos ir metrikos. Biblioteką sudaro 296 uždaviniai `AMPL`, `GAMS`, `JuMP` ir `Pyomo` skaliarinio modelio formatu. Biblioteka buvo sukurta naudojant pavyzdinius modelius, esančius `GAMS` modelių bibliotekoje. Prie kiekvieno modelio pateikiamas optimizavimo problemos tipas, lygčių, kintamųjų, diskrečiųjų kintamųjų, nulinių elementų ir netiesinių nulinių elementų skaičius. Be to, vartotojas gali rasti visą modelio konvertavimo metu surinktą statistiką, išsamią ir standartinę `GAMS CONVERT` įrankio išvestį.

Toliau paaiškinami įrankiai ir procesai, skirti sukurti biblioteką nuo nulio. Pateikiamas autoriaus sukurtas automatinis įrankis `gamslib-convert.sh`, kuris geba sugeneruoti AMK testavimo biblioteką. Įrankis yra laisvai prieinamas *GitHub* saugykloje ir geba ne tik konvertuoti pateiktus `GAMS` modelius į kitus AMK, bet ir surinkti bei dokumentuoti visas modelio charakteristikas, nustatytas konvertavimo proceso metu. Įrankis palaiko du skirtingus vykdymo režimus: masinį visų modelių bibliotekos generavimą ir vieno modelio generavimą.

Vėliau pristatomos išvados apie įrankių, naudotų kuriant biblioteką, kokybę. 35 modelių nepavyko konvertuoti pilnai licencijuotu `GAMS CONVERT` įrankiu dėl vykdymo arba kompiliavimo klaidų. Tai reiškia, kad kai kurie `GAMS` bibliotekos modeliai nesuderinami su pačia `GAMS` modeliavimo sistema. Vėliau, atliekant modelio egzempliorių kūrimo testą, nustatyta, kad 12 `AMPL`, 11 `JuMP` ir 29 `Pyomo` modelių, sugeneruotų `GAMS CONVERT` įrankiu, buvo klaidų. Daugumą `Pyomo` ir `JuMP` klaidų sukėlė neteisingas `GAMS CONVERT` įrankio veikimas, kai buvo sukurtas klaidingas *Suffix* primityvumo apibrėžimas.

Vėliau pristatomos išvados apie įrankių, naudotų kuriant biblioteką, kokybę. 35 modelių nepavyko konvertuoti licencijuotu `GAMS CONVERT`

įrankiu dėl vykdymo arba kompiliavimo klaidų. Tai reiškia, kad kai kurie GAMS bibliotekos modeliai nesuderinami su pačia GAMS modeliavimo sistema. Vėliau, testuojant modelio egzempliorių kūrimą, nustatyta, kad 12 AMPL, 11 JuMP ir 29 Pyomo modeliuose, sugeneruotuose GAMS CONVERT įrankiu, buvo klaidų. Daugumą Pyomo ir JuMP klaidų lėmė netinkamai veikiantis GAMS CONVERT įrankis, kai buvo sukurtas klaidingas Suffix primityvumo apibrėžimas.

Toliau buvo išnagrinėta klasikinė transportavimo problema, modeliuojama skirtingomis AMK. Palyginimas atliktas remiantis šiais kriterijais: modelio dydis baitais, modelio dydis pagal kodo eilučių skaičių, modelio dydis pagal naudojamų kalbos primityvų skaičių, modelio egzemplioriaus sukūrimo laikas. Nustatyta, kad modeliai, parengti AMPL, GAMS ir JuMP, yra kompaktiškiausi, o modelis, parengtas Pyomo, yra didžiausias. Palyginus modeliui sukurti reikalingų kalbos primityvų skaičių, JuMP ir AMPL parodė geriausius rezultatus. Tai gali reikšti, kad šios modeliavimo kalbos gali turėti švelnesnę mokymosi kreivę, o tai taip pat leidžia daryti išvadą, kad nagrinėjamų algebrinio modeliavimo kalbų kontekste JuMP ir AMPL leidžia praktikams suformuluoti optimizavimo problemą glausčiausiai. Išnagrinėjus transporto modelio egzemplioriaus kūrimo laiką pastebėta, kad AMPL išsiskiria iš kitų AMK ir yra labiausiai optimizuota našumo požiūriu. Menki JuMP našumo rezultatai patvirtino Dunning et al. teiginius, kad JuMP sistemos pirminis paleidimo laikas yra pastebimai ilgas.

Vėliau buvo pristatyti didelio masto modelio egzempliorių kūrimo tyrimo rezultatai. Tyrime buvo matuojamas laikas, per kurį modeliavimo sistema atlieka modelio egzempliorių kūrimo ir eksportavimo operacijas. Buvo panaudoti visi modeliai, esantys anksčiau sukurtoje testavimo bibliotekoje. Taip pat pateikiamas `load-benchmark.sh` įrankis, pasiekiamas autorių *GitHub* saugykloje. Jis įkelia kiekvieną modelį į tam tikrą modeliavimo sistemą, tuomet eksportuoja jį į sprendėjams suprantamą formatą, fiksuoja vykdymo statistiką ir generuoja palyginimo ataskaitą. Šis įrankis yra laisvai prieinamas ir kitiems tyrėjams. Nustatyta ta pati tendencija, kaip ir transporto problemos modelio testuose. AMPL buvo neabejotinai geriausias, o JuMP ir Pyomo pasirodė prasčiausiai. Taip pat pastebėta, kad vidutinis skirtumas tarp

AMPL ir kitų varžovų didėja, kai modeliai tampa didesni. Nebuvo rasta reikšmingų skirtumų tarp skirtingų optimizavimo problemų tipų, išskyrus JuMP, kur modelio egzemplioriaus sukūrimo laikas skiriasi dirbant su skirtingų tipų problemomis. Be to, skirtumai tarp skirtingų to paties tipo modelių taip pat yra reikšmingesni naudojant JuMP kalbą. Galima manyti, kad tai lemia dinamiška Julia programavimo kalbos, kuria sukurtas JuMP, prigimtis bei JuMP savybė saugoti sukurtus modelius laikinojoje talpykloje. Galima daryti išvadą, kad iš nagrinėtų AMK AMPL yra akivaizdžiai našiausia AMK modelio egzemplioriaus kūrimo metu.

Po to JuMP tyrimas, kurį atliko JuMP autoriai [18], lygintas su šio darbo našumo tyrimais. Nustatyta, kad kai kurios tendencijos išlieka, tačiau šio darbo JuMP testų ir Dunning et al. testų rezultatai labai skiriasi. Taigi buvo nuspręsta palyginti testų metodiką ir rezultatus atliekant identišką testą, kuris pateikiamas Dunning et al. tyrime. Šio darbo JuMP testas patvirtino pastebėjimą, kad JuMP kenkia ilgas darbo pradžios laikas, reikalingas iš anksto kompiliuoti JuMP bibliotekas. Autoriui nepavyko pasiekti JuMP našumo metrikos, apie kurią pranešė Dunning et al., kai JuMP visada pranoksta Pyomo. Rastus skirtumus gali lemti skirtingos naudojamos JuMP versijos, patobulintas Pyomo našumas arba skirtingos Gurobi sprendėjo versijos.

Galiausiai, siekiant nustatyti, kokią praktinę naudą sukuria AMPL išankstinis sprendimas sprendžiant optimizavimo uždavinius, buvo atliktas testas, įvertinantis AMPL išankstinį sprendimą. Atlikus testavimą su 286 modeliais, esančiais testavimo bibliotekoje, pastebėta, kad AMPL išankstinis sprendėjas sugebėjo supaprastinti modelius 52,8 % atvejų, iš kurių 5 kartus iš 7 galėjo nustatyti, kad problemos sprendimas neegzistuoja, taip net nereikalaujant iškviesti tikrojo sprendėjo. Vidutiniškai pritaikius AMPL išankstinį sprendimą pavyko sumažinti modelio dydį pašalinant 18,42 % apribojimų ir 10,73 % kintamųjų. Siekiant įvertinti, ar AMPL išankstinis sprendimas iš tikrųjų daro teigiamą įtaką problemų sprendimui, buvo atliktas papildomas testas. AMPL presolve darė teigiamą įtaką 26,43 % atvejų iteracijų ir 47,86 % laiko atžvilgiu. Tačiau tai pat padarė neigiamą poveikį 20,71 % atvejų iteracijų ir 23,57 % atvejų laiko atžvilgiu. Vis tik galima daryti išvadą, kad teigiamas poveikis visada yra reikšmingesnis už neigiamą ir jis išryškėja,

kai sprendėjas neturi arba nenaudoja vidinių problemų išankstinio sprendimo mechanizmų.

Algebrinių modeliavimo kalbų skirtumai ir trūkumai

Šiame skyriuje pateikiami pagrindiniai tyrimo metu nustatyti populiariausių AMK skirtumai ir trūkumai.

Pirmiausia buvo pristatytos bendrosios funkcijos, sintaksė ir jų suderinamumo iššūkiai. Nustatyta, kad AMK vartotojai negali būti lankstūs, kai reikia pereiti nuo vienos AMK prie kitos ir panaudoti anksčiau įgytas žinias. Tai gali sąlygoti nepageidautiną prisirišimą prie vieno gamintojo AMK. Be to, kadangi darbo pradžioje ne visada aišku, su kokio tipo problema susiduriama, praktikai gali pasirinkti netinkamą AMK, nepalaikančią tam tikro problemos tipo. Taip pat gali kilti situacijų, kai turimas sprendėjas veikia tik kitoje AMK, nei yra aprašytas modelis, ir vartotojas juo negali pasinaudoti.

Toliau buvo pristatyti našumo skirtumai ir sudėtingesnės savybės, kaip, pavyzdžiui, lygiagretinimas. Eksperimentiniame tyrime buvo nustatyta, kad skirtingų modeliavimo aplinkų našumas ženkliai skiriasi, todėl pravartu būti lankstiems renkantis geriausią AMK dideliems uždaviniams spręsti arba tais atvejais, kai reikalingos papildomos funkcijos, tokios kaip išankstinis sprendimas ar lygiagretinimas.

Galiausiai, pateikiama įžvalgų santrauka, leidžianti daryti išvadą, kad nors yra keletas pajėgių modeliavimo aplinkų ir AMK, nė viena iš jų nesuteikia viso funkcijų rinkinio, reikalingo efektyviam ir intuityviam matematinio optimizavimo problemų sprendimui. Sudėtingumas pereinant nuo vienos AMK prie kitos ir poreikis išmokti konkrečios AMK sintaksę gali tapti iššūkiu mokant matematinio optimizavimo mokyklose ir universitetuose.

Universali optimizavimo sistema

Šiame skyriuje nusakyti reikalavimai ir pateiktas pasiūlymas universaliai optimizavimo sistemai, kurią sudaro WebAML kalba ir optimizavimo sistemos prototipas.

Siūloma formali kalba įgalina aprašyti problemos charakteristikas JSON formatu, o prototipas leidžia sukurti modelį naudojant WebAML kalbą ir jį išspręsti naudojant jau egzistuojančias AMK. Šiuo metu WebAML kalba palaiko tik pagrindines funkcijas, tačiau, remiantis JSON schema, ji yra lanksčiai plečiama. Darbe yra pateikti pavyzdžiai, kaip tokie plėtiniai galėtų būti realizuoti.

Atvirojo kodo žiniatinklyje veikiantis prototipas buvo sukurtas kaip įrankis, leidžiantis naudotojui sukurti modelį, konvertuoti jį į konkretų AMK formatą ir pateikti jį spręsti lokaliam ar nuotoliniam sprendėjui (pvz., NEOS serveriui). Tokiu būdu ne kuriamas naujas uždavinio sprendimo funkcionalumas, o remiamasi geriausiomis ypatybėmis, kurias teikia populiariausios AMK. Architektūriniai sprendimai, priimti kuriant prototipą, sudaro gerą pagrindą universaliam matematiniam optimizavimo įrankių rinkiniui.

Autorius mano, kad pavyko pasiūlyti universalią optimizavimo sistemą, kuri nereikalauja jokių specifinių algebrinės kalbos žinių ir leidžia spręsti problemas naudojant skirtingus matematinius optimizavimo sprendėjus. Tai supaprastina algebrinio modeliavimo ir matematinio optimizavimo procesą: taip jis tampa prieinamas asmenims, neturintiems išsamių techninių žinių. Dėl to jis patrauklus ne tik komerciniams vartotojams, bet ir mokytojams, dėstytojams ir studentams, bandantiems suprasti matematinio optimizavimo pagrindus. Visame pasaulyje atlikti informacinių ir ryšių technologijų (IRT) tyrimai parodė, kad IKT gali padėti pagerinti mokinių mokymąsi ir padėti parinkti geresnius mokymo metodus (pvz., [61]).

Įrankis taip pat palaiko visus pagrindiniuose AMK esančius sprendėjus, taigi suteikia daug nemokamų ir komercinių sprendėjų. Kadangi įrankis gali būti lengvai įdiegtas serveryje ir pasiekiamas per žiniatinklio sąsają, mokslo įstaiga gali įsigyti akademinę ar komercinę sprendėjo licenciją ir sudaryti sąlygas kiekvienam bendruomenės nariui lengvai spręsti didelės apimties optimizavimo problemas.

Bendros išvados

1. Šiame tyrime pasiūlyta universalios optimizavimo sistemos koncepcija, kurią sudaro formalizuota algebrinio modeliavimo kalba (WebAML) ir atvirojo kodo įrankis (WebAML optimizavimo sistema), skirtas algebriniam modeliavimui ir matematiniam optimizavimui.

   (a) Šia sistema siekiama sukurti patogią aplinką, supaprastinančią matematinį modeliavimą ir optimizavimą, tačiau išlaikančią geriausias pagrindinių AMK savybes.

   (b) Sistema palaiko visus sprendėjus, kuriuos palaiko ir pagrindinės AMK, taigi praktikai gali pasirinkti iš plataus spektro sprendėjų. Tai leidžia praktikams lengvai eksperimentuoti sprendžiant įvairių tipų problemas ir rasti geriausią įrankį konkrečiam uždaviniui spręsti.

   (c) Sistema gali konvertuoti modelį iš vieno AMK į kitą. Tokiu būdu praktikai gali lengvai migruoti tarp skirtingų AMK ir naudoti tą, kuri teikia daugiausia funkcijų arba yra našiausia.

   (d) Įrankis nereikalauja jokių specifinių algebrinės kalbos žinių ir leidžia spręsti problemas naudojant skirtingus AMK ir optimizavimo sprendėjus. Tokiu būdu sudaromos sąlygos ne tik lengviau išmokti ir mokyti matematinio optimizavimo, bet leisti pažengusiam specialistui spręsti realias matematines optimizavimo problemas.

2. Siekiant įrodyti tokios koncepcijos įgyvendinamumą, sukurtas universalios optimizavimo sistemos prototipas, realizuojantis pagrindines siūlomos koncepcijos ypatybes. Tuo pačiu yra pateikiami aiškūs plėtimo taškai ir idėjos, kaip tokį įrankį būtų galima tobulinti toliau.

   (a) Siūloma WebAML kalba gali būti išplėsta siekiant dar labiau supaprastinti modelio apibrėžimo procesą. Tą galima pasiekti patobulinus vartotojo sąsają (pvz., pridedant tekstinių nurodymų)

ir praplečiant kalbos sintaksę (pvz., palaikant neišreikštinį aibių apibrėžimą).

(b) Prototipas gali būti išplėstas taip, kad būtų palaikomos papildomos funkcijos, pavyzdžiui, lygiagretus modelio generavimas ar išankstinis sprendimas. Galima įtraukti daugiau AMK kuriant naujus konvertavimo modulius iš WebAML į tikslinį AMK. Visa tai galima pasiekti panaudojant prototipe jau apibrėžtas modulių sąsajas ir įtraukiant naujus modulius pasitelkus priklausomybių įterpimo konfigūraciją.

(c) Tyrimus ir vystymą galima tęsti siekiant WebAML optimizavimo sistemoje realizuoti automatizuoto sprendimo algoritmo pasirinkimą, o tai leistų ženkliai supaprastinti specialistų darbą.

3. Analizuojant universalios optimizavimo sistemos reikalavimus buvo atlikta esamų AMK eksperimentinė analizė ir tyrimai, kurių tikslas buvo įvertinti šias AMK charakteristikas: modelio dydį ir išraiškingumą, modelio egzemplioriaus sukūrimo laiką, išankstinį sprendimą ir jo poveikį sprendimui.

   (a) Modeliai, aprašyti `AMPL`, `GAMS` ir `JuMP`, yra kompaktiškiausi, o `Pyomo` aprašytas modelis yra didžiausias. Palyginus modeliui sukurti reikalingų kalbos primityvų skaičių, `JuMP` ir `AMPL` parodė geriausius rezultatus. Tai gali reikšti, kad šios modeliavimo kalbos gali turėti švelnesnę mokymosi kreivę, o tai taip pat leidžia daryti išvadą, kad apžvelgtų algebrinio modeliavimo kalbų kontekste `JuMP` ir `AMPL` leidžia glausčiausiai suformuluoti optimizavimo problemą.

   (b) Modelio egzempliorių kūrimo našumo testas nustatė, kad `AMPL` yra greičiausias, o `JuMP` ir `Pyomo` – lėčiausi. Nėra reikšmingų skirtumų tarp skirtingų optimizavimo problemų tipų, išskyrus `JuMP`, kai modelio egzemplioriaus sukūrimo laikas ženkliai skiriasi dirbant su skirtingų tipų problemomis.

   (c) Našumo skirtumas tarp `AMPL` ir kitų varžovų didėja, kai modeliai tampa didesni. Lyginant didelių modelių (modelių, turinčių

daugiau nei 500 lygčių, testavimo bibliotekoje yra 8 tokie modeliai) egzempliorių kūrimo laiką nustatyta, kad skirtumas tarp AMPL ir GAMS yra 11 kartų didesnis, o tarp AMPL ir Pyomo – 38 kartus didesnis. Skirtumas tarp AMPL ir JuMP yra beveik 100 kartų didesnis, kai skirtumas tarp GAMS ir Pyomo išliko maždaug toks pat – apie 3,5 karto.

(d) JuMP našumo tyrimo rezultatai patvirtina Dunning et al. teiginį, kad JuMP paleisties sąnaudos yra pastebimos net ir mažiausiems uždaviniams spręsti. Šio darbo autoriaus testuose vien tik paketo JuMP inicijavimas užtruko apie 7 sekundes. Taip pat pastebėtas reikšmingas pagreitėjimas kuriant kelis JuMP modelio egzempliorius iš eilės. Taigi tai gali būti priimtina, kai modelių šeima per vieną sesiją išsprendžiama kelis kartus, o kompiliavimo išlaidos patiriamos tik pirmą kartą išsprendus egzempliorių.

(e) AMPL, būdama vienintelė AML, palaikanti išankstinį sprendimą, 52,8 % atvejų supaprastino modelius, iš kurių 5 kartus iš 7 galėjo nustatyti, kad problema neturi sprendinio, taip net nereikalaujant iškviesti sprendėjo. Vidutiniškai pritaikius AMPL išankstinį sprendimą pavyko sumažinti modelio dydį pašalinant 18,42 % apribojimų ir 10,73 % kintamųjų. AMPL išankstinis sprendimas padarė teigiamą įtaką 26,43 % atvejų iteracijų ir 47,86 % atvejų laiko atžvilgiu. Tačiau tai padarė neigiamą poveikį 20,71 % atvejų iteracijų ir 23,57 % atvejų laiko atžvilgiu. Visgi teigiamas poveikis yra didesnis nei neigiamas, ypač tais atvejais, kai sprendėjas neturi problemų sprendimo algoritmų.

Trumpos žinios apie disertantą

Vaidas Jusevičius gimė 1986 m. gegužės 20 d. Vilniuje. 2005 m. baigė Vilniaus Žemynos gimnaziją. Vilniaus Universiteto Matematikos ir Informatikos fakultete įgijo informatikos bakalauro laipsnį (2009 m.) ir informatikos magistro laipsnį (2011 m.). 2017 – 2021 m. buvo Vilniaus Universiteto doktorantas. Nuo 2012 m. dėsto Vilniaus Universiteto Matematikos ir Informatikos fakulteto Programų sistemų katedroje.

ACKNOWLEDGMENTS

I want to express my deepest appreciation to my scientific supervisor Prof. Dr. Remigijus Paulavičius, as this work would not have been possible without his guidance and support.

I am grateful to both dissertation reviewers, Doc. Dr. Algirdas Lančinskas and Prof. Dr. Dmitrij Šešok, who carefully read the dissertation and provided valuable advice and critical remarks, which helped improve the manuscript's final quality.

I sincerely thank my parents Rima and Vitalijus for their support, even in the most challenging moments, and my colleagues Mónica and Sophia who always motivated me to move forward.

PUBLICATIONS BY THE AUTHOR

Articles in the reviewed scientific periodical publications:

1. **Vaidas Jusevičius** and Remigijus Paulavičius. ”Web-Based Tool for Algebraic Modeling and Mathematical Optimization”. Mathematics, 2021, 9 (21), 2751. DOI: 10.3390/math9212751.

2. **Vaidas Jusevičius**, Richard Oberdieck and Remigijus Paulavičius. ”Experimental Analysis of Algebraic Modelling Languages for Mathematical Optimization”. Informatica, 2021, 32 (2), 283–304. DOI: 10.15388/21-INFOR447.

NOTES

NOTES

Vaidas Jusevičius

ATVIROJO KODO ALGEBRINIO MODELIAVIMO IR MATEMATINIO OPTIMIZAVIMO SISTEMOS KŪRIMAS IR TYRIMAS

Daktaro disertacijos santrauka
Gamtos mokslai
Informatika (N 009)
Redaktorė Rūta Anusevičienė

Vaidas Jusevičius

RESEARCH AND DEVELOPMENT OF AN OPEN-SOURCE ALGEBRAIC MODELING AND MATHEMATICAL OPTIMIZATION SYSTEM

Doctoral Dissertation
Natural Sciences
Informatics (N 009)
Editor Vydas Geidrichis

Vilniaus universiteto leidykla
Saulėtekio al. 9, III rūmai, LT-10222 Vilnius
El. p. info@leidykla.vu.lt, www.leidykla.vu.lt
bookshop.vu.lt, journals.vu.lt
Tiražas 20 egz.