Certified Evaluation for LLMs in Optimization Modeling: From Graph Isomorphism to Formulation Isomorphism

Zhuohan Wang [*1] **Ziwei Zhu** [*1] **Ziniu Li** [1] **Congliang Chen** [2 3] **Zhihang Lin** [1 4] **Mingzhe Yang** [1] **Yizhou Han** [1] **Yufeng Lin** [1] **Angyang Gu** [1] **Xinglin Hu** [1] **Ruoyu Sun** [1 5] **Tian Ding** [2 4 5]

Abstract

Large language model (LLM) benchmarks typically evaluate correctness through task-specific rules, but for structured outputs, evaluation requires a precise notion of formulation-level correspondence. We study this issue in automated optimization modeling, where LLMs translate natural-language descriptions and numerical data into solver-ready formulations. Existing solver-based evaluation only approximates modeling correctness: it offers no formulation-level guarantee, is uninformative for infeasible cases, and can be expensive on hard mixed-integer linear programs. To address these limitations, we propose ORGEval, a graph-theoretic evaluation framework that represents canonical optimization formulations as bipartite graphs and reduces equivalence checking to graph isomorphism. To make this practical and reliable, we introduce symmetric decomposability (SD), a checkable condition under which Weisfeiler-Lehman color equivalence is exactly equivalent to formulation isomorphism. ORGEval combines bipartite WL testing with efficient SD verification to provide provably correct equivalence evaluation for SD-certified instances. We further construct Bench4Opt, a benchmark that separates models from data and contains SD-certified ground-truth instances. Experiments show that ORGEval reliably detects formulation equivalence while being substantially faster than solver-based evaluation, especially on challenging instances. Benchmarking state-of-the-art LLMs reveals that optimization modeling remains difficult, with the best model achieving only 57.11% accuracy.

1. Introduction

Large language models (LLMs) have rapidly evolved into powerful general-purpose problem-solving systems, demonstrating strong performance across professional and academic exams, mathematical reasoning, code generation, and increasingly complex domain-specific tasks (Achiam et al., 2023; Guo et al., 2025; Jimenez et al., 2023). Such progress has made benchmarking central to the development and deployment of foundation models: benchmarks are now used to compare models, diagnose capabilities, expose limitations, guide post-training, and support claims about model reliability. Recent evaluation efforts such as BIG-bench and HELM have substantially expanded the scope of LLM evaluation, covering broader task families, model behaviors, and desiderata such as robustness, calibration, fairness, bias, toxicity, and efficiency (Suzgun et al., 2023; Bommasani et al., 2023). Yet many benchmarks remain largely empirical: they report aggregate scores without formal guarantees about what the evaluation oracle certifies. This limitation is especially important for structured outputs, where correctness is often an equivalence problem rather than surface matching or agreement on a single observed outcome.

Automated optimization modeling provides a concrete example of this challenge. Operations Research (OR) uses mathematical models to support decision-making in logistics, manufacturing, finance, healthcare, and many other domains (Hillier & Lieberman, 2015; Winston, 2004). A central task in OR practice is to translate a natural-language problem description and associated data into a solver-ready formulation. This process requires both domain understanding and optimization expertise: one must identify decision variables, formulate the objective, encode constraints, and ensure that the resulting model faithfully represents the intended decision problem. Recent work has explored the use of LLMs to automate this modeling process, including generating mathematical formulations and executable solver code from natural-language specifications (Jiang et al., 2024; Huang et al., 2025; Lu et al., 2025). How-

[1]The Chinese University of Hong Kong (Shenzhen), Shenzhen, China [2]Shenzhen Research Institute of Big Data, Shenzhen, China [3]Shenzhen Loop Area Institute, Shenzhen, China [4]AutoKernel, Shenzhen, China [5]Shenzhen International Center for Industrial and Applied Mathematics, Shenzhen, China. Correspondence to: Tian Ding <dingtian@sribd.cn>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

ever, reliably evaluating these outputs remains difficult because correct optimization formulations are not syntactically unique. Two equivalent models may differ in variable names, constraint orderings, or indexing conventions, while two non-equivalent models may produce the same optimal objective value on a particular data instance. Therefore, evaluating optimization modeling requires an equivalence notion that is invariant to superficial re-indexing but sensitive to structural modeling errors.

Existing OR modeling benchmarks commonly rely on solver-based evaluation: the generated and reference formulations are solved under a fixed data configuration, and their optimal objective values are compared (Tang et al., 2024; Huang et al., 2024; AhmadiTeshnizi et al., 2024; Yang et al., 2024b). Such an evaluation strategy provides no formal guarantee of formulation correctness. Non-equivalent models can coincidentally share the same optimal value; infeasible instances make objective-value comparison uninformative; repeated solver calls can be costly for large or difficult mixed interger programming (MILPs). These limitations illustrate a broader benchmark-design problem: an empirical evaluation protocol may be easy to run, but without a theory of what the evaluation certifies, its scores can be misleading.

In this work, we propose a theory-informed evaluation framework for optimization modeling by formalizing optimization-modeling correctness as structural equivalence. We say two LP/MILP formulations are equivalent if their decision variables and constraints can be permuted so that objective coefficients, constraint coefficients, right-hand sides, constraint senses, and variable types match exactly. This definition is invariant to superficial renaming and reordering while remaining sensitive to modeling errors. It also leads naturally to a graph-theoretic representation: each optimization instance is encoded as a weighted bipartite graph whose node partitions represent constraints and variables, with edge weights and node features encoding the coefficients and modeling metadata. Under this representation, formulation equivalence reduces to graph isomorphism. This reduction makes the evaluation problem mathematically explicit: the correctness of a generated model becomes the isomorphism between two structured graphs.

However, graph isomorphism (GI) testing is a nontrivial problem. The Weisfeiler-Lehman (WL) test is an efficient and widely used heuristic for GI, but it is not complete in general: two non-isomorphic graphs can receive identical WL color distributions (Cai et al., 1992). Directly using WL would therefore yield another heuristic evaluator without correctness guarantees. To address this issue, we identify a sufficient condition, which we call *symmetric decomposability* (SD), under which the WL test is guaranteed to decide isomorphism correctly for the bipartite optimization graphs. We further provide an efficient procedure for verifying whether a given instance satisfies SD.

The SD condition serves both a theoretical and practical role. Theoretically, it identifies a regime in which efficient WL-based evaluation is provably correct. Practically, it guides benchmark construction: all ground-truth instances in our benchmark are pre-verified to satisfy SD. Since SD is preserved under isomorphism, any generated instance that is not SD can be safely rejected as non-equivalent; if it is SD, WL color comparison becomes a provably correct equivalence test. Thus, the benchmark contains not only tasks and labels, but also a checkable condition certifying when the evaluation oracle is reliable.

Building on this theory, we introduce ORGEVAL, a graph-theoretic framework for evaluating LLM-generated LP/MILP formulations, and BENCH4OPT, a model-data separated benchmark for optimization modeling. The model-data separated design reflects real OR practice, where reusable model code is maintained separately from numerical data files. It also avoids a common limitation of prior benchmarks that embed all numerical data directly in the prompt, thereby restricting problem scale and conflating modeling ability with instance-specific arithmetic. BENCH4OPT contains 197 LP and MILP problems, each expressed at two abstraction levels, yielding 394 word problems in total. Each example includes a natural-language problem description, a parameter file, and reference solver code that instantiates the ground-truth optimization instance.

Our framework connects theory, benchmark construction, and empirical analysis. Theory informs the evaluation oracle through the graph-isomorphism reduction and the SD correctness condition. The benchmark is then constructed and verified so that its ground-truth instances fall within the certified evaluation regime. Finally, empirical results obtained through this benchmark reveal concrete capability limits and failure modes of current LLMs in optimization modeling. In our experiments, ORGEval achieves consistent structural evaluation across data configurations and is substantially faster than solver-based evaluation on difficult instances. When applied to state-of-the-art LLMs, Bench4Opt shows that automated optimization modeling remains challenging, and error analysis indicates that constraint specification and decision-variable design remain dominant failure modes. We defer a detailed comparison with prior work on OR modeling benchmarks to Appendix A.

Our contributions are summarized as follows.

- We formalize optimization-modeling correctness for as formulation isomorphism and reduce LP/MILP equivalence evaluation to graph isomorphism (Section 3.1 and Section 3.3).
- We introduce SD, prove that WL-based comparison is complete for SD bipartite optimization graphs, and

show that random instantiation yields SD with high probability under mild assumptions (Section 3.2 and appendix C.4).
- We construct Bench4Opt, a model-data separated benchmark of 394 optimization modeling tasks whose ground-truth instances are SD-certified (Section 4.1).
- We benchmark leading LLMs and identify structural failure modes, especially in constraint specification and decision-variable design (Section 4.2 and Section 4.3).

2. Background and Problem Formulation

2.1. Background

In this work, we focus on two classes of optimization problems: LP and MILP. Both classes have broad applications across domains such as finance, supply chain management, transportation, and wireless communications.

An LP/MILP problem, denoted by $\mathcal{P}$, admits the following standard formulation:[1]

$$\mathcal{P}: \quad \min_{\mathbf{x}\in\mathbb{R}^p\times\mathbb{Z}^{n-p}} \mathbf{c}^\top\mathbf{x} \quad \text{s.t. } \mathbf{A}\mathbf{x}\circ\mathbf{b}, \tag{1}$$

where $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\mathbf{c}\in\mathbb{R}^n$, $\mathbf{b}\in\mathbb{R}^m$, $\circ\in\{=,<,>,\leq,\geq\}^m$ for $i=1,\cdots,n$; for subsequent use, we employ a vector τ to represent the decision variable type, where $\tau_i=1$ indicates that x_i is an integer variable, and $\tau_i=0$ indicates a continuous variable.

An optimization modeling task for LP/MILP involves translating a textual problem description, together with the associated numerical data, into a solver-readable instance of the form given in Equation (1).

In practice, reusable model code is often maintained separately from numerical data, as in AMPL and Pyomo (Fourer et al., 2003; Hart et al., 2011). This model-data separation is central to our setting: an LLM-generated formulation should define variables, objectives, and constraints in a way that can be instantiated under different data configurations.

2.2. Problem Formulation

In this section, we formalize metrics for modeling correctness. Establishing such metrics requires a precise definition of a mathematical model, which we characterize as a mapping from problem data to concrete optimization instances.

2.2.1. Definitions for Modeling Problems

In the context of automated modeling discussed in Section 2.1, a model does not correspond to a single optimization instance such as $\mathcal{P}$ in Equation (1). Instead, it induces a family of instances via a mapping from an admissible data space to the space of optimization problem instances.

Definition 2.1 (MILP/LP Model)**.** A MILP/LP model is a mapping $\mathcal{M}:\Theta\to\mathcal{P}_{A,b,c,\circ,\tau}$, where Θ denotes a space of problem data and $\mathcal{P}$ is a model instance with parameters $(A,b,c,\circ,\tau)$. Here, Θ may be any subset of $\mathbb{R}^q$ representing the admissible data of the modeling family, with q denoting the dimension of the problem data.

In practice, part of Θ is fixed as structural components, while the remaining part consists of real-valued elements that vary within a closed subset of $\mathbb{R}$. We refer to Θ as the **problem data support** of $\mathcal{M}$. Given any $\theta\in\Theta$, the mapping $\mathcal{M}(\theta)$ returns a concrete MILP or LP instance $\mathcal{P}$ of the form Equation (1). An illustrative example of a model and its problem data support is provided in Theorem D.4.

2.2.2. Solver-Based Modeling Correctness

The dominant approach in prior work evaluates modeling correctness through execution correctness (Tang et al., 2024; Huang et al., 2024; AhmadiTeshnizi et al., 2024; Yang et al., 2024b), which we formalize as follows:

Definition 2.2 (Execution Correctness)**.** Given a data configuration θ, suppose that the ground-truth model $M^*(\theta)$ has a finite optimal objective value $f^*(\theta)$. A mathematical model $\mathcal{M}$ with program code $\mathcal{C}$ is said to be *execution-correct* on θ if executing $\mathcal{C}$ on θ terminates successfully and returns an objective value $z(\mathcal{C},\theta)$ such that

$$z(\mathcal{C},\theta)=f^*(\theta).$$

Execution correctness is intuitive but only result-based: non-equivalent formulations may share the same optimal value (Theorem D.1), infeasible cases yield no comparable objective value (Theorem D.3), and repeated solver calls inherit the computational cost of solving hard MILPs. We therefore use it only as a baseline and define the target correctness relation structurally.

2.2.3. Structure-Based Modeling Correctness

As an alternative to solver-based evaluation, we define **formulation isomorphism** to capture structural equivalence between formulations. Intuitively, two formulations are considered equivalent if there exists a one-to-one correspondence between their variables (and between their constraints) under which the objective, constraints, and all coefficients are equivalent. We formalize this as follows.

Definition 2.3 (Formulation Isomorphism)**.** For a data configuration θ, a model instance $\mathcal{P}=\mathcal{M}(\theta)$ is said to be isomorphic to the ground-truth instance $\mathcal{P}^*=\mathcal{M}^*(\theta)$ if its components, as defined in Equation (1), share the same algebraic structure as those of the ground truth. Formally,

[1] In Equation (1), the formulation involves p continuous decision variables, $n-p$ integer decision variables, and m constraints. If $p=n$, i.e., there are no integer decision variables, the problem reduces to an LP. Otherwise, it is an MILP.

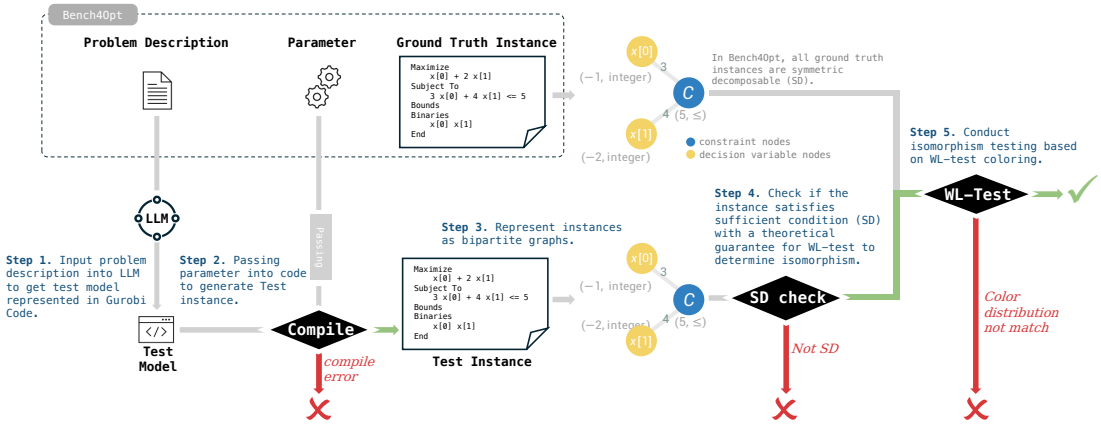

Figure 1. Evaluation pipeline. Each dataset example includes a problem description, data file, and solver-ready reference code. Using the same data configuration, we compare instances by representing them as bipartite graphs and applying isomorphism testing with sufficiency-condition verification.

there must exist permutation matrices P_1 and P_2 such that $\mathcal{P}$ can be transformed into $\mathcal{P}^*$ via

$$\begin{aligned} \tau^* &= P_1\tau, \quad \mathbf{c}^* = P_1\mathbf{c}, \quad \mathbf{b}^* = P_2\mathbf{b}, \\ \mathbf{A}^* &= P_2\mathbf{A}P_1^\top, \quad \circ^* = P_2 \circ . \end{aligned} \tag{2}$$

where $P_1 \in \{0,1\}^{n\times n}$ and $P_2 \in \{0,1\}^{m\times m}$ represent the reordering of decision variables and constraints.

Throughout this work, we refer to this notion as isomorphism equivalence, or simply equivalence. We write $\mathcal{P}_1 \sim \mathcal{P}_2$ to denote that two instances $\mathcal{P}_1$ and $\mathcal{P}_2$ are isomorphic. In practice, we evaluate equivalence by testing formulation isomorphism on a given data configuration.

Remark. Formulation isomorphism is intended to evaluate faithful reconstruction of a given canonical formulation, rather than arbitrary optimality-preserving or efficiency-oriented reformulations. In our setting, two formulations are regarded as equivalent only when they preserve the algebraic correspondence between variables, constraints, coefficients, and the original problem description, up to permissible renaming and reordering. This evaluation target is therefore narrower than the broader notion of modeling equivalence commonly used in optimization practice. For example, adding slack variables, aggregating or disaggregating constraints, introducing lifted variables, applying presolve reductions, or using alternative linearizations may preserve the optimal value, but they change the structural correspondence between the generated formulation and the canonical one.

3. Methodology

In this section, we present ORGEval, our proposed evaluation method, along with its theoretical guarantees. ORGEval evaluates modeling equivalence by comparing the structural representations of two instances derived from two models.

3.1. Model Equivalence Based on Graph Isomorphism

Our notion of model equivalence naturally aligns with graph isomorphism, which allows nodes to be re-indexed or rearranged without altering the graph structure. This motivates us to leverage graph-theoretic tools for equivalence evaluation. Following prior work on learning to optimize (Gasse et al., 2019; Chen et al., 2022b), we represent each LP/MILP instance as a bipartite graph (see Figure 8). We show that equivalence detection can be reduced to graph isomorphism testing; see Appendix C.4 for a formal proof.

Building on this, we propose a graph-based framework that evaluates modeling results via the following procedure:

Create test and standard graphs Following the notation of Chen et al. (2022b), we represent (MI)LP instances as bipartite graphs:

Definition 3.1 (Weighted Bipartite Graph Instance Representation)**.** A MILP/LP instance can be represented as a weighted bipartite graph $\mathbf{G} = (\mathbf{V} \cup \mathbf{W}, \mathbf{E})$, where $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ corresponds to constraints and $\mathbf{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ corresponds to variables. An edge (v_i, w_j) is present if and only if $\mathbf{A}_{ij} \neq 0$, and its weight is $\mathbf{A}_{ij}$. Each constraint node is connected to the variable nodes that appear in that constraint. Each vertex is associated with features (e.g., right-hand side b_i, operator type $\circ_i$ for constraints; objective coefficient c_j, integrality type τ_j for variables).

As described in Definition 3.1, the resulting bipartite graphs partition nodes into two disjoint sets (variable nodes and constraint nodes), each equipped with the relevant features. Given these graph representations, equivalence between formulations can be detected via graph isomorphism testing.

Isomorphism testing Graph isomorphism testing is a long-standing open problem with no known polynomial-time algorithm (Garey & Johnson, 1979; Babai, 2016). The

WL test (Leman & Weisfeiler, 1968) is a computationally efficient heuristic for graph isomorphism testing. If the WL test produces different color distributions for two graphs, the graphs are guaranteed to be non-isomorphic. However, if the WL test yields identical color distributions, the two graphs are not necessarily isomorphic (Cai et al., 1992); see Appendix D for counterexamples.

Unlike the standard WL test, which does not guarantee correct isomorphism detection for all graphs, our enhanced algorithm first verifies that a sufficient condition is satisfied. This verification step provides a formal correctness guarantee for the subsequent equivalence evaluation.

3.2. Sufficient Condition for Graph Isomorphism Testing

We propose a sufficient condition, termed *symmetric decomposability*, under which modeling instances can be reliably tested for isomorphism by a polynomial-time algorithm.

Definition 3.2 (Symmetric Decomposable Instance)**.** We say a modeling instance $\mathcal{P}$ is **symmetric decomposable** if, after WL-test, the coloring on its representation graph $\mathcal{G}$ satisfies the following conditions: Excluding nodes that are uniquely colored, the remaining nodes can be divided into k disjoint groups (with some $k \geq 0$) of the same size, denoted by $S_1, S_2, \cdots, S_k$, where

1. All nodes in the same group have distinct colors,
2. All groups share the same coloring sets, and
3. Every two groups are disconnected, i.e. $\forall$ nodes $a \in S_i, b \in S_j, i \neq j$, a is disconnected with b.

SD strictly generalizes the unfoldable condition of Chen et al. (2022b): unfoldable instances have only singleton WL color classes, whereas SD also allows repeated color classes when they decompose into disconnected color-identical blocks. This covers common MILP structures such as bin-packing instances; examples can be found in Theorem D.7, Figure 11, Figure 12, and Figure 13.

In the following theorem, we show that if both instances are SD, then identical WL color distributions are equivalent to isomorphism, and Algorithm 1 correctly determines whether a test instance is equivalent to the standard instance. A rigorous proof is provided in Appendix C.7.

Theorem 3.3. *Suppose $\mathcal{P}_1$ and $\mathcal{P}_2$ are symmetric decomposable. Then $\mathcal{G}_1$ and $\mathcal{G}_2$ share the same coloring distribution after WL-test coloring $\Longleftrightarrow \mathcal{P}_1 \sim \mathcal{P}_2$.*

To apply Theorem 3.3 in practice, we design an algorithm to identify symmetric decomposable instances (Algorithm 3). Moreover, we prove that under mild assumptions, randomly sampling from the problem data space yields a symmetric decomposable instance with high probability (Theorems C.11 and C.12). Leveraging this result, we construct a benchmark dataset in which all ground-truth instances are guaranteed to be symmetric decomposable.

Remark (Why Algorithm 1 only checks the test instance). Since SD is invariant under graph isomorphism and all Bench4Opt ground-truth instances are pre-verified as SD (see Section 4.1), ORGEval only checks whether the test instance is SD or not. A non-SD test instance is immediately rejected; otherwise, equivalence is decided by comparing WL color multisets.

Algorithm 1 Modeling Equivalence Detection

Require: Two graph instances $(G_k, H_k) \in \mathcal{G}^k_{m,n} \times \mathcal{H}^V_m \times \mathcal{H}^W_n$ and adjacency matrix $\mathbf{A}_k, k = 1, 2$; iterate limit $L > 0$.
1: Color nodes in two graphs using WL-test Algorithm for MILP/LP, get two coloring multi-sets $\mathcal{C}_k = \left\{ \{\{C_i^{k,V}\}\}_{i=0}^m, \{\{C_j^{k,W}\}\}_{j=0}^n \right\}, k = 1, 2$ for coloring $\mathcal{G}_1$ and $\mathcal{G}_2$.
2: Derive set of unique elements in $\mathcal{C}_k$, denote as set $\mathbb{A}_k, \forall k = 1, 2$.
3: **if** $\mathcal{C}_1 \neq \mathcal{C}_2$ **then**
4: **return** Not equivalent
5: **else if** $\mathcal{G}_2$ is symmetric decomposable **then**
6: **return** Equivalent
7: **else**
8: **return** Not Equivalent

3.3. ORGEval: Model-Equivalence Evaluation Based on Graph Isomorphism

Combining the symmetric decomposability detection algorithm (Algorithm 3) with the WL test, we develop ORGEval, an augmented WL-test procedure for evaluating model equivalence. ORGEval can provably determine whether a test instance is equivalent to a symmetric decomposable ground-truth instance. Given a reference instance and a generated instance, ORGEval builds their bipartite graphs, runs WL on both[2], and applies the following rule: if the WL color multisets differ, return non-equivalent; if the generated graph is not SD, return non-equivalent; otherwise return equivalent.

Evaluation efficiency. For an instance with m constraints, n variables, and k SD clusters, ORGEval runs in $\mathcal{O}(kmn)$ time, avoiding the $\mathcal{O}(n!)$ cost of exhaustive isomorphism testing; see Section C.9.

4. Benchmark Construction and Empirical Analysis

A benchmark for optimization modeling should not only rank models, but also make explicit when its evaluation decisions are reliable. Our empirical study is therefore organized around three questions. First, can the theoretical guarantee

[2]We use the same implementation of WL test as (Chen et al., 2022b), presented as Algorithm 2.

developed in Section 3 be operationalized as a benchmark-design principle. Second, once the benchmark instances are certified, does the resulting evaluation oracle provide advantages over solver-based evaluation in efficiency and stability? Third, what do the certified benchmark results reveal about the current capabilities and failure modes of LLMs in optimization modeling?

4.1. Bench4Opt: A Certified Model–Data Separated Benchmark

We introduce BENCH4OPT, a benchmark for evaluating LLM-generated LP/MILP formulations under a model–data separated paradigm. This design is motivated by real-world OR practice, where the reusable formulation is typically maintained separately from numerical data. Modeling languages such as AMPL and Pyomo similarly distinguish model files from data files (Fourer et al., 2003; Hart et al., 2011). In contrast, many existing optimization-modeling benchmarks place both the problem description and all numerical data directly in the prompt (Ramamonjison et al., 2022a; Xiao et al., 2023), which restricts problem scale, lengthen prompts, and conflates modeling ability with instance-specific arithmetic.

Each example in BENCH4OPT contains three components:

- **Problem description** (`wp.txt`). This file describes the optimization scenario, objective, constraints, and required parameters in natural language. It does not embed concrete numerical values, but specifies the structure of the data that will be provided separately.
- **Parameter file** (`parameter.json`). This file contains the numerical data used to instantiate the model. Separating this file from the problem statement allows the same formulation to be evaluated under different data configurations.
- **Reference model** (`code.txt`). This file contains expert-verified Python/Gurobi code that reads the parameter file and generates the corresponding LP/MILP instance, saved as an `.lp` file for evaluation.

This format evaluates a modeling system at the level of reusable formulations rather than single-instance solutions. Given a problem description, the LLM must generate solver-ready code that correctly defines decision variables, objective, and constraints, while reading numerical data from the provided parameter file. ORGEval instantiates the generated and reference code with the same parameters and checks structural equivalence of the resulting instances.

BENCH4OPT contains 197 LP and MILP problems. Each problem is expressed at two abstraction levels, yielding 394 word problems in total. The *structured* version follows a standardized template with explicit sections such as background, problem description, parameters, decision variables, objective, constraints, implementation notes, and expected outcome. The *unstructured* version is a concise natural-language description that removes much of this scaffolding. This two-level design allows us to test both component-level modeling ability under explicit guidance and more realistic modeling ability under less structured descriptions. Detailed problem class and size coverage can be found in Appendix B

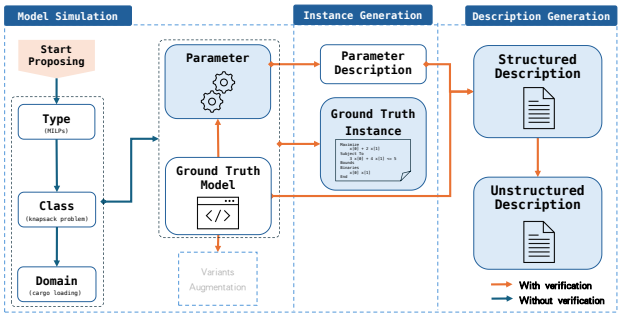

Figure 2. Data Construction Pipeline: Word problems are constructed through hierarchical reverse data evolution with model simulation and description generation. Processes with verification by either LLMs or human experts are marked in orange.

Hierarchical reverse data evolution. To construct diverse problems with controlled quality, we use a hierarchical reverse data evolution pipeline, see Figure 2.. The pipeline proceeds from optimization models to data files and then to natural-language word problems. We start from two types of seed problems. The first type consists of LLM-proposed seed models. For these, GPT-4o is used to propose classical LP/MILP problem classes, select application domains, and generate canonical optimization models. These models are then diversified by varying objectives, constraints, decision variables, and domain-specific contexts. The second type consists of seed instances selected from MIPLIB (Gleixner et al., 2021). For MIPLIB-derived seeds, OR experts translate selected instances into abstract Gurobi model code and data-generation code. We then use GPT-4o to produce domain-specific adaptations and variants.

After the seed models are constructed, we generate reference answers and parameter files. For MIPLIB-derived problems, domain experts annotate data-generation code so that multiple parameter files can be produced for the same formulation. For LLM-generated seed models, GPT-4o generates Python programs and associated `parameter.json` files, which are then executed and checked. Finally, we reverse-generate word problems from the verified solver code. Inspired by the INFORMS AIMMS-MOPTA modeling competition (AIMMS, 2024), we first generate structured descriptions following a standardized template, and then summarize them into concise unstructured descriptions.

Quality control. Because benchmark reliability depends on both task quality and evaluation correctness, we apply multiple layers of validation. First, we use a controlled code

skeleton to constrain the generation of reference models. This turns open-ended code generation into a structured completion problem and improves consistency across examples. Second, all generated reference programs are executed, and only programs that successfully produce valid LP/MILP instances are retained. Third, during word-problem generation, an LLM-based verifier checks whether the textual description aligns with the objective, variables, constraints, and parameter structure in the reference code. We further involve four PhD students with OR expertise for human verification. Details about human-expert verification process is stated in Appendix C.3.

SD certification. The most important distinction between BENCH4OPT and conventional empirical benchmarks is that its ground-truth instances are certified for evaluation reliability. After constructing the benchmark, we apply the SD detection procedure in Algorithm 3 to the bipartite representation of every ground-truth instance. All ground-truth instances underlying BENCH4OPT satisfy the SD condition. We did not use SD as a filtering criterion when proposing seed problems; instead, SD verification is applied as a certification step after benchmark construction.

4.2. Efficiency and Reliability of the Evaluation Oracle

Before using ORGEval to benchmark LLMs, we first validate the evaluation oracle itself. This step is essential because our central claim is not simply that ORGEval produces different scores from solver-based evaluation, but that it produces evaluation decisions with a formal reliability guarantee on certified instances.

Evaluation efficiency. We compare the runtime of ORGEval with solver-based evaluation on MIPLIB instances across three difficulty levels: easy, hard, and open. We sample 75 instances in total, with 25 instances from each difficulty level. Following the MIPLIB categorization, easy instances can typically be solved within one hour on standard hardware, hard instances require substantially more computation or specialized techniques, and open instances remain unsolved. The results are shown in Table 1.

Solver-based evaluation inherits the difficulty of the instance and becomes unavailable for open cases. In contrast, ORGEval evaluates structural equivalence directly and does not solve the optimization problem. This demonstrates that structural evaluation can remain tractable precisely in regimes where solver-based evaluation becomes expensive or unavailable.

Evaluation consistency across data configurations. A reusable optimization model should remain correct across data configurations, not only under a single numerical instantiation. The model–data separated design of BENCH4OPT allows us to test this property directly. For each problem, we instantiate the same reference and generated model under five randomly sampled parameter configurations and compare whether the evaluation result remains stable. The result are shown in Table 2.

Table 1. Evaluation time comparison between ORGEval and solver-based methods across three difficulty levels (easy, hard, open). All instances are sampled from MIPLIB, with 25 instances per level.

Difficulty	Avg. Size (#vars + #cons.)	Avg. Eval Time (Solver)	Avg. Eval Time (ORGEval)
Easy	1,848	about 1 hour	**0.21 s**
Hard	10,463	$>$ 1 hour	**3.83 s**
Open	17,050	not yet solved	**32.07 s**

Table 2. Comparison of evaluation consistency across five random data configurations under different evaluation methods. The table reports the proportion of problems for which each evaluation method yields consistent results. Feasibility consistency denotes the fraction of problems for which all five instances are feasible under the solver.

Model	Feasibility consistency	ORGEval consistency	Solver consistency
GPT-4o	36.30%	100.00%	95.58%
Claude-Opus-4	36.05%	100.00%	92.12%
DeepSeek-V3	34.69%	100.00%	93.95%
OpenAI o1	35.43%	100.00%	94.77%
Average	35.62%	100.00%	94.11%

ORGEval achieves 100% consistency across all five configurations for all evaluated problems, as shown in Table 2. This is expected from the design of ORGEval: it evaluates the algebraic structure induced by the model and is not affected by incidental solver behavior. Solver-based evaluation is much less stable. Even conditional on feasibility, solver-based decisions are not perfectly stable, with an average consistency of 94.11%. These results show that objective-value comparison can depend strongly on the chosen data configuration, whereas ORGEval provides a stable structural evaluation.

Together, the efficiency and consistency experiments validate the role of SD-certified structural evaluation as a benchmark oracle. The benchmark score is not merely the result of running a heuristic or a solver on a single instance; it is produced by an evaluation procedure whose correctness is characterized by a checkable condition.

4.3. Benchmarking LLM Optimization Modeling

Having validated the evaluation oracle, we use ORGEval and BENCH4OPT to benchmark state-of-the-art LLMs on optimization modeling. All models are evaluated under direct prompting. The main results are reported in Table 3. The results show that automated optimization modeling remains challenging even for leading LLMs. Current LLMs are far from reliably translating natural-language OR problems into structurally correct LP/MILP formulations. Moreover, from the structured/unstructured breakdown, models perform consistently better on structured descriptions than on unstructured descriptions. This gap suggests that current

LLMs benefit substantially from explicit decomposition of the modeling task into variables, objectives, constraints, and implementation notes. When this scaffolding is removed, models struggle more with recovering the implicit structure of the optimization problem.

Table 3. Evaluation Results on Bench4Opt. We report pass@1 accuracy for each LLM. The Overall columns summarize results across the full Bench4Opt dataset. The Breakdown columns report accuracy at two levels of abstraction: structured and unstructured. Examples of each problem type are provided in Figure 5 and Figure 4. Best scores within each category are highlighted in red.

LLMs	Overall		Breakdown	
	Acc. (%)	Comp. Err. (%)	Struct. (%)	Unstruct. (%)
Claude-Opus-4-7	57.11	1.27	61.93	52.28
DeepSeek-V3	54.82	2.28	63.45	46.19
GPT-4.1	52.28	1.78	57.36	47.21
GPT-4o	51.02	7.36	58.38	43.65
OpenAI o3	47.97	0.76	55.84	40.10
DeepSeek-R1	47.72	2.28	55.84	39.59
OpenAI o1	47.21	1.78	52.79	41.62

Failure analysis. To better understand the empirical findings, we conduct a focused error analysis on GPT-4o and OpenAI o1. We annotate failed outputs using two broad categories. Compilation errors include Python syntax or semantic errors, Gurobi API misuse, and misunderstandings of the problem/data interface. Modeling errors include wrong objectives, wrong constraints, and wrong decision variables. Modeling-error labels are not mutually exclusive, since a single generated formulation may contain multiple structural mistakes. The result is reported in Table 7.

For GPT-4o, failed outputs include both compilation-level issues and structural modeling mistakes, indicating that the model may struggle not only with generating executable optimization code but also with correctly translating problem semantics into mathematical formulations. Among modeling errors, incorrect constraints are the most frequent, while objective-function errors occur less often.

For OpenAI o1, the failed cases are more often associated with structural modeling errors than with low-level executability issues. In particular, constraint specification remains a major source of failure, and errors in decision-variable design also appear in some cases. Objective-function errors are relatively uncommon. These observations suggest that, even when generated code is executable, faithfully reconstructing the structural semantics of an optimization problem remains challenging.

Overall, the analysis highlights that the main bottleneck in optimization modeling lies in capturing the correct formulation structure, especially constraints and decision variables. This also illustrates the value of a structural benchmark: solver-based evaluation or compile-error rates alone may obscure these failure modes, while ORGEval exposes them through a reliable equivalence criterion.

Table 4. Error analysis on Bench4Opt. Each percentage reports the fraction of *failed outputs* from a given model that exhibit the corresponding error type. Modeling-error categories are not mutually exclusive, so the percentages do not sum to 100.

Error Type	GPT-4o	o1
Python syntax / semantics	9.97	0.00
Gurobi API misuse	5.30	0.40
Problem/data-interface misunderstanding	4.36	1.61
Wrong objective	3.43	3.23
Wrong constraints	56.07	60.08
Wrong decision variables	27.41	39.92

5. Conclusion

We presented ORGEval as a theory-informed evaluation framework for LLM-generated LP/MILP formulations. Motivated by the lack of formal guarantees in solver-based evaluation, we replaced objective-value comparison with a structural notion of formulation equivalence. By representing optimization instances as weighted bipartite graphs, we reduced formulation equivalence to graph isomorphism and introduced symmetric decomposability (SD), a checkable sufficient condition under which WL-based graph comparison is provably correct. Thus, ORGEval not only evaluates generated formulations, but also characterizes when the evaluation decision itself is reliable.

We further introduced Bench4Opt, a model–data separated benchmark containing 197 LP/MILP problems and 394 word problems across two abstraction levels. All ground-truth instances in Bench4Opt are verified to satisfy the SD condition, making the benchmark a certified evaluation setting rather than only a collection of tasks and reference answers. Experiments show that ORGEval achieves 100% consistency across data configurations and evaluates hard MIPLIB instances within seconds, while solver-based evaluation can become unstable, uninformative, or computationally prohibitive. Benchmarking state-of-the-art LLMs on Bench4Opt reveals that optimization modeling remains difficult: the best models achieve only 57.11% pass@1 accuracy, and error analysis shows that constraint specification and decision-variable design are the dominant failure modes.

ORGEval connects theory, benchmark design, and empirical analysis: graph isomorphism and SD provide a reliable evaluation oracle, Bench4Opt instantiates it with SD-certified examples, and the results expose current LLM limitations. This highlights a broader principle for structured-domain benchmarks: evaluation should specify not only model scores, but also when the oracle is provably reliable.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

AhmadiTeshnizi, A., Gao, W., and Udell, M. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024.

AIMMS. 16th aimms-mopta optimization modeling competition. https://coral.ise.lehigh.edu/ mopta/competition, 2024.

Anastacio, M. and Hoos, H. H. Combining sequential model-based algorithm consingh2012overviewuration with default-guided probabilistic sampling. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pp. 301–302, 2020.

Ansótegui, C., Sellmann, M., and Tierney, K. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pp. 142–157. Springer, 2009.

Astorga, N., Liu, T., Xiao, Y., and van der Schaar, M. Autoformulation of mathematical optimization models using llms. *arXiv preprint arXiv:2411.01679*, 2024.

Babai, L. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 684–697, 2016.

Bergman, D., Huang, T., Brooks, P., Lodi, A., and Raghunathan, A. U. Janos: an integrated predictive and prescriptive modeling framework. *INFORMS Journal on Computing*, 34(2):807–816, 2022.

Bobrow, D. G. A question-answering system for high school algebra word problems. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, pp. 591–614, 1964.

Bommasani, R., Liang, P., and Lee, T. Holistic evaluation of language models. *Annals of the New York Academy of Sciences*, 1525(1):140–146, 2023.

Cai, J., Kadioglu, S., and Dilkina, B. Gala: Global llm agents for text-to-model translation. *arXiv preprint arXiv:2509.08970*, 2025.

Cai, J.-Y., Fürer, M., and Immerman, N. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

Chen, H., Constante-Flores, G. E., Mantri, K. S. I., Kompalli, S. M., Ahluwalia, A. S., and Li, C. Optichat: Bridging optimization models and practitioners with large language models. *INFORMS Journal on Data Science*, 2025.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022a.

Chen, Z., Liu, J., Wang, X., Lu, J., and Yin, W. On representing linear programs by graph neural networks. *arXiv preprint arXiv:2209.12288*, 2022b.

Chi, C., Aboussalah, A., Khalil, E., Wang, J., and Sherkat-Masoumi, Z. A deep reinforcement learning framework for column generation. *Advances in Neural Information Processing Systems*, 35:9633–9644, 2022.

Dakle, P. P., Kadıoğlu, S., Uppuluri, K., Politi, R., Raghavan, P., Rallabandi, S., and Srinivasamurthy, R. Ner4opt: Named entity recognition for optimization modelling from natural language. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 299–319. Springer, 2023.

Dellarosa, D. A computer simulation of children's arithmetic word-problem solving. *Behavior Research Methods, Instruments, & Computers*, 18(2):147–154, 1986.

Elmachtoub, A. N. and Grigas, P. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.

Fourer, R., Gay, D. M., and Kernighan, B. W. Specifying data. In *AMPL: A Modeling Language for Mathematical Programming*. Thomson/Brooks/Cole, 2nd edition, 2003. URL https://ampl.com/wp-content/uploads/Chapter-9-Specifying-Data-AMPL-Book.pdf. Chapter 9.

Garey, M. R. and Johnson, D. S. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Gleixner, A., Hendel, G., Gamrath, G., Achterberg, T., Bastubbe, M., Berthold, T., Christophel, P. M., Jarck, K., Koch, T., Linderoth, J., Lübbecke, M., Mittelmann, H. D., Ozyurt, D., Ralphs, T. K., Salvagnin, D., and Shinano, Y. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi: 10.1007/s12532-020-00194-3. URL https://doi.org/10.1007/s12532-020-00194-3.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hart, W. E., Watson, J.-P., and Woodruff, D. L. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

Hillier, F. S. and Lieberman, G. J. *Introduction to operations research*. McGraw-Hill, 2015.

Huang, C., Tang, Z., Hu, S., Jiang, R., Zheng, X., Ge, D., Wang, B., and Wang, Z. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 2025.

Huang, X., Shen, Q., Hu, Y., Gao, A., and Wang, B. Mamo: a mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144*, 2024.

Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., and Yu, Y. Llmopt: Learning to define and solve general optimization problems from scratch. *arXiv preprint arXiv:2410.13213*, 2024.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Leman, A. and Weisfeiler, B. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

Liu, F., Yang, Z., Liu, C., Song, T., Gao, X., and Liu, H. Mm-agent: Llm as agents for real-world mathematical modeling problem. *arXiv preprint arXiv:2505.14148*, 2025.

Liu, H., Zheng, Z., Qiao, Y., Duan, H., Fei, Z., Zhou, F., Zhang, W., Zhang, S., Lin, D., and Chen, K. Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark. *arXiv preprint arXiv:2405.12209*, 2024.

Lu, H., Xie, Z., Wu, Y., Ren, C., Chen, Y., and Wen, Z. Optmath: A scalable bidirectional data synthesis framework for optimization modeling. *arXiv preprint arXiv:2502.11102*, 2025.

Maragno, D., Wiberg, H., Bertsimas, D., Birbil, Ş. İ., den Hertog, D., and Fajemisin, A. O. Mixed-integer optimization with constraint learning. *Operations Research*, 2023.

Michailidis, K., Tsouros, D., and Guns, T. Constraint modelling with llms using in-context learning. In *30th International conference on principles and practice of constraint programming*, 2024.

Rajgopal, J. Principles and applications of operations research. *Maynard's Industrial Engineering Handbook.–2004.–P*, pp. 11–27, 2004.

Ramamonjison, R., Li, H., Yu, T. T., He, S., Rengan, V., Banitalebi-Dehkordi, A., Zhou, Z., and Zhang, Y. Augmenting operations research with auto-formulation of optimization models from problem descriptions. *arXiv preprint arXiv:2209.15565*, 2022a.

Ramamonjison, R., Yu, T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., and Zhang, Y. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In Ciccone, M., Stolovitzky, G., and Albrecht, J. (eds.), *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 28 Nov–09 Dec 2022b. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

Sawada, T., Paleka, D., Havrilla, A., Tadepalli, P., Vidas, P., Kranias, A., Nay, J. J., Gupta, K., and Komatsuzaki, A. Arb: Advanced reasoning benchmark for large language models. *arXiv preprint arXiv:2307.13692*, 2023.

Singirikonda, A., Kadioglu, S., and Uppuluri, K. Text2zinc: A cross-domain dataset for modeling optimization and satisfaction problems in minizinc. *arXiv preprint arXiv:2503.10642*, 2025.

Sundaram, S. S. and Khemani, D. Natural language processing for solving simple word problems. In *Proceedings of the 12th international conference on natural language processing*, pp. 394–402, 2015.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q., Chi, E., Zhou, D., et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Talbi, E.-G. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

Tang, Z., Huang, C., Zheng, X., Hu, S., Wang, Z., Ge, D., and Wang, B. Orlm: Training large language models for optimization modeling. *arXiv preprint arXiv:2405.17743*, 2024.

Tsouros, D., Verhaeghe, H., Kadıoğlu, S., and Guns, T. Holy grail 2.0: From natural language to constraint models. *arXiv preprint arXiv:2308.01589*, 2023.

Wang, P.-W., Donti, P., Wilder, B., and Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pp. 6545–6554. PMLR, 2019.

Winston, W. L. *Operations research: applications and algorithm*. Thomson Learning, Inc., 2004.

Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., et al. Chain-of-experts: When llms meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2023.

Yang, Z., Huang, Y., Shi, W., Feng, L., Song, L., Wang, Y., Liang, X., and Tang, J. Benchmarking llms for optimization modeling and enhancing reasoning via reverse socratic synthesis. *arXiv preprint arXiv:2407.09887*, 2024a.

Yang, Z., Wang, Y., Huang, Y., Guo, Z., Shi, W., Han, X., Feng, L., Song, L., Liang, X., and Tang, J. Optibench meets resocratic: Measure and improve llms for optimization modeling. *arXiv preprint arXiv:2407.09887*, 2024b.

Zeng, S., Kody, A., Kim, Y., Kim, K., and Molzahn, D. K. A reinforcement learning approach to parameter selection for distributed optimal power flow. *Electric Power Systems Research*, 212:108546, 2022.

Zhai, H., Lawless, C., Vitercik, E., and Leqi, L. Equivamap: Leveraging llms for automatic equivalence checking of optimization formulations. *arXiv preprint arXiv:2502.14760*, 2025.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Zhou, Z., Liu, S., Ning, M., Liu, W., Wang, J., Wong, D. F., Huang, X., Wang, Q., and Huang, K. Is your model really a good math reasoner? evaluating mathematical reasoning with checklist. *arXiv preprint arXiv:2407.08733*, 2024.

A. Related Work

NLP for OR Modeling While substantial progress has been made in automatic modeling of general mathematical problems (Bobrow, 1964; Dellarosa, 1986; Sundaram & Khemani, 2015; Liu et al., 2025), work targeting operations research (OR) modeling has only recently begun to accelerate. Prior to the rise of LLMs, the NL4Opt competition (Ramamonjison et al., 2022b) explored the feasibility of learning-based natural language interfaces for optimization solvers. With the advent of LLMs, several studies have demonstrated their potential as modeling assistants. Multi-agent and decomposition-based approaches—such as Holy Grail 2.0 (Tsouros et al., 2023), Chain-of-Experts (CoE) (Xiao et al., 2023), OptiMUS (AhmadiTeshnizi et al., 2024), OptiChat (Chen et al., 2025), and GALA (Cai et al., 2025). Similarly, focuses on bridging practitioner workflows with optimization models by enabling LLM-based interpretation, debugging, and reformulation. Complementary work such as NER4OPT (Dakle et al., 2023) and Text2Zinc (Singirikonda et al., 2025) investigates front-end components and datasets to structure natural language into entities, constraints, and MiniZinc specifications, thereby supporting more reliable text-to-model translation. Recent work on in-context constraint modeling (Michailidis et al., 2024) further demonstrates that even general-purpose LLMs can produce executable constraint programs using retrieval and few-shot prompting alone. In parallel, (Tang et al., 2024) showed that fine-tuning mid-size open-source models can yield substantial modeling improvements, underscoring the growing importance of domain-adapted OR modeling systems.

Given the increasing use of LLMs for OR modeling, there is a strong need for benchmarks that reveal their capability boundaries (Liu et al., 2024; Zhou et al., 2024; Sawada et al., 2023). Several optimization modeling datasets have been proposed. The Linear Programming Word Problem (LPWP) dataset (Ramamonjison et al., 2022a) spans multiple domains but focuses primarily on elementary-level LP problems. ComplexOR (Xiao et al., 2023) introduces more sophisticated scenarios, yet its limited size and reliance on numeric data embedded within text still restrict the modeling complexity it captures. Other datasets—including IndustryOR (Tang et al., 2024), MAMO (Huang et al., 2024), and E-OPT (Yang et al., 2024a)—use synthesis and augmentation to broaden coverage. The NLP4LP dataset (AhmadiTeshnizi et al., 2024) attempts to separate numeric data from textual descriptions, but its problem scale remains small, and the descriptions are relatively structured with explicit variable/constraint/objective declarations. In the constraint programming community, Text2Zinc (Singirikonda et al., 2025) introduces a cross-domain MiniZinc benchmark that also follows a model–data separated design, separating natural-language problem descriptions and parameter specifications from concrete data instances. It is a valuable resource for studying general CP/OR text-to-model translation. However, the paper reports primarily automated verification procedures and does not describe full manual validation for all included instances. Compared to these benchmarks, our work aims to provide both a more comprehensive dataset and a more rigorous evaluation methodology, enabling a more precise assessment of LLM capabilities in optimization modeling.

Modeling Equivalence Evaluation The earliest work to evaluate NLP for OR modeling performance is to calculate the canonical accuracy (Ramamonjison et al., 2022a). This accuracy counts for the declaration-level(e.g., objective or constraints) matching score between predicted and reference formulations. This method has severe limitations as it's highly sensitive to superficial differences in formulation, such as variable naming or ordering. More recent benchmark works—including MAMO (Huang et al., 2024), IndustryOR (Tang et al., 2024), NLP4LP (AhmadiTeshnizi et al., 2024), and OptiBench (Yang et al., 2024b)—relies on solvers to assess modeling quality. They execute the predicted numerical models and compare the resulting optimal values with reference optimal values to evaluate correctness. While solver-based approaches better align with the functional goals of optimization, they introduce new limitations. The evaluation becomes dependent on solver behavior, which is often unstable, especially when compared model instances coincidentally share the same solution, are infeasible, or are hard to solve. As a result, optimal value mismatches may stem not from modeling errors but from solver or numerical issues, thereby confounding the reliability of equivalence assessment. Beyond optimal-value–based equivalence evaluation, (Astorga et al., 2024) use satisfiability modulo theories (SMT) to assess formulation equivalence. SMT operates by encoding formulations into logical constraints and checking their satisfiability under the feasible region of the decision variables. However, such methods only work when the correspondence between decision variables is already provided, whereas identifying a correct mapping is often one of the most challenging parts of comparing two formulations. (Zhai et al., 2025) attempt to prompt an LLM to infer a mapping between the decision variables of two formulations, followed by a solver-based verification step. If a valid mapping is found, the formulations are equivalent. While this offers a valuable perspective beyond objective-based evaluation, two limitations remain. First, LLM-generated mappings introduce computational overhead and may be unreliable, with hallucinated mappings or missing potential mappings. Second, the final verification still depends on a solver, which cannot reliably handle infeasible instances and may incur substantial computation when solvers struggle.

Broader Research on AI for OR Beyond model formulation, significant progress has been made in the field of AI for Operations Research (AI for OR), particularly in parameter generation and solving optimization problems (Rajgopal, 2004). In parameter generation, AI techniques have been employed for better simulation of key parameters of optimization problems (Elmachtoub & Grigas, 2022; Maragno et al., 2023; Bergman et al., 2022). Similarly, our work leverages LLMs to generate necessary problem data through a program of thoughts (Chen et al., 2022a). On the optimization side, numerous studies have focused on leveraging AI models in automatic algorithm configuration (Ansótegui et al., 2009; Lindauer et al., 2022; Anastacio & Hoos, 2020), optimization algorithm selection (Wang et al., 2019; Chi et al., 2022), and heuristic algorithm design (Zeng et al., 2022; Talbi, 2009; Romera-Paredes et al., 2024). Specifically, a line of research has modeled MILP/LP problems as bipartite graphs and applied Graph Neural Networks (GNNs) to make decisions at various stages of their solution processes (Gasse et al., 2019; Zhou et al., 2020). These GNN-based methods have demonstrated efficacy in tasks such as variable selection and node branching, leading to significant improvements in solver performance. Inspired by this, we model optimization problems as bipartite graphs and formalize the evaluation paradigm based on the classical WL-test algorithm (Leman & Weisfeiler, 1968).

B. Dataset

B.1. Data Format and Structure

Inspired by the INFORMS modeling competition (AIMMS, 2024), we adopt a model-parameter-separated format to emulate real-world optimization modeling tasks in our word problems. This mirrors standard practices in industries, where problem formulation and parameter collection are typically sequential processes handled by distinct teams.

Each word problem (WP) is structured as follows:

- **Problem Description (‘wp.txt‘)**: A comprehensive description of the optimization scenario, including objectives and constraints, without embedding specific parameter values. It also specifies the nature and structure of the required data, guiding the LLM on the information needed without providing actual values.
- **Parameter File (‘parameter.json‘)**: A structured file containing all the parameter values necessary to model and solve the problem. This separation ensures that LLMs formulate the problem based on the description before applying the parameter values.
- **Reference Model (‘code.txt‘)**: A reference code to the modeling problem in Python Gurobi code. Reading the associated parameter file will produce the corresponding .lp instance (model.lp) for evaluation.

An illustrative example is provided in Figure 2, demonstrating how the problem description and parameter file complement each other. Examples of our word problems are listed in Figure 4 and Figure 5 in the Appendix. By decoupling problem complexity from parameter dimension, we can create conceptually challenging problems without imposing an overly long prompt on LLMs.

B.2. Data construction

To efficiently construct multidimensional complexity, we employ a hierarchical reverse data evolution pipeline comprising three key stages: optimization model stimulation, reference answer generation, and word problem generation, see Figure 2.

Optimization Models Generation Our dataset is constructed from two categories of seed problems: 1) seed problems proposed by an LLM, and 2) seed problems selected from MIPLIB. For the LLM-proposed seed problems, we employ GPT-4o to systematically generate optimization models by progressively expanding the problem context across types, classes, domains, and variants. GPT-4o first identifies classical problem classes within linear programming (LP) and mixed-integer linear programming (MILP). For each class, it then selects representative application domains to ensure practical relevance. Within each domain–problem-class pair, GPT-4o produces a canonical optimization model and subsequently enriches it by varying core modeling components—objectives, constraints, and decision variables—to introduce structured diversity and increased complexity. In addition, we incorporate seed problems from the MIPLIB dataset. Operations research expert translates the selected MIPLIB instances into abstracted Gurobi model code along with the necessary data files. Because directly converting from the original LP files is often challenging, we restrict our selection to relatively small instances; the largest contains 48 variables and 34 constraints. We then use GPT-4o to rewrite these MIPLIB-derived models by generating domain-specific adaptations and variant formulations.

Reference Answers Generation For problems augmented from MIPLIB seed problem, we ask domain experts to annotate the corresponding data-generation code, enabling multiple generations of associated parameter files (parameter.json) . For problems augmented from LLM-generated seed instances, we used GPT-4o to automatically produce Python programs tailored to each optimization model and to generate the required parameter.json files. This procedure could also be helpful if variation in problem size

is needed by modifying dimensionality specifications directly in the prompts, thereby streamlining the expansion of individual optimization models. During evaluation, each simulated optimization model reads its associated parameter file and produces the corresponding .lp instance (model.lp).

Word Problems Generation Finally, we reversely generate word problems from the stimulated optimization models. Drawing inspiration from the INFORMS AIMMS-MOPTA (AIMMS, 2024) Optimization Modeling Competition, we meticulously designed a standardized word problem structure, as illustrated in Figure 2. Using GPT-4o, we first translate the solver code into detailed word problems adhering to this standardized structure. Subsequently, we refine and summarize the generated content to produce concise and unstructured problem statements. This approach ensures that the generated word problems accurately represent the underlying optimization models while being suitable for benchmarking in a complex and realistic manner.

B.3. Quality control

To pursue high data quality in our benchmark, we employ a controlled generation framework with systematic verification.

Controlled Generation Similar to the design of standardized word problem structures, we develop a structured code skeleton tailored to guide optimization models. This approach transforms an open-ended generation task into a constrained code completion task, enabling tight regulation of LLM output. By partially automating the data evolution and verification pipeline, our framework ensures consistency across generated instances and adherence to predefined structural requirements, as illustrated in Figure 6.

Verification We systematically validate each model-generated answer pair. During model simulation and answer generation, we execute the corresponding code representations, retaining only those that run without errors. Additionally, in the word problem generation stage, we leverage an LLM-based verifier to confirm the precise alignment between optimization components in the code and their textual descriptions. To further enhance reliability, we involved four PhD students with expertise in OR in the verification process. The verification consists of two stages:

- **Alignment Verification** To ensure a strict alignment between each word problem and its associated model code. Specifically, all objectives, decision variables, and constraints described in the word problem must be accurately reflected in the model code, with no omissions or extraneous elements. All generated problems were reviewed by the experts. Minor issues (e.g., typographical errors or minor inconsistencies) were corrected. Problems exhibiting major mismatches—such as missing or incorrect objectives, variables, or constraints—were discarded.

- **Representative Verification** To verify that the reference model codes align with the word problems. A random sample of 50 word problems was selected. Experts constructed their own optimization model based on the word problem descriptions, without reference to the original code. The resulting models were then compared to the reference implementations. The experts' answers for all 50 word problems match the reference answer.

By integrating controlled generation with rigorous verification, we uphold high standards of data quality and accuracy. This ensures the reliability and robustness of our benchmark for evaluating large language models in operations research modeling.

Table 5. Optimization problem types and classes covered in our Bench4Opt.

Problem Type	Problem Class
8*LPs	Diet Problem
	Transportation Problem
	Blending Problem
	Production Planning Problem
	Network Flow Problem
	Portfolio Optimization Problem
	Cutting Stock Problem
	Staff Scheduling Problem
5*MILPs	Knapsack Problem
	Bin Packing Problem
	Capacitated Facility Location Problem
	Capital Budgeting Problem
	Assignment Problem

B.4. Dataset Coverage

Table 6. Industry domain covered in our Bench4Opt.

Category	Representative Application Scenarios
Transportation, Routing & Logistics	Vehicle routing, cargo/vehicle loading, postal and parcel delivery, food delivery, waste collection, public transportation planning, school bus routing, logistics and distribution, transportation network design.
Supply Chain, Production & Manufacturing	Supply chain management, production planning, manufacturing scheduling, agricultural/food/pharmaceutical distribution, industrial material flow, chemical/metal/glass/paper/textile manufacturing.
Facility Location & Spatial Planning	Warehouse location optimization, facility location and assignment, fire station placement, healthcare facility location, wildlife reserve planning, public infrastructure placement, districting and territory design.
Workforce, Scheduling & Operational Planning	Employee rostering, hospital nurse scheduling, call center scheduling, university professor scheduling, public transport driver shifts, construction worker scheduling, classroom scheduling, school timetabling, project task assignment.
Resource Allocation, Budgeting & Decision Optimization	Portfolio optimization, investment planning, advertising budget allocation, general resource allocation, R&D portfolio planning, energy distribution and generation, telecommunications network design and frequency allocation.
Agriculture, Environment & Public Services	Agricultural land use planning, animal feed formulation, fertilizer optimization, environmental conservation, water and wastewater management, emergency services deployment, healthcare services, nutrition planning for hospitals, schools, and elderly care.

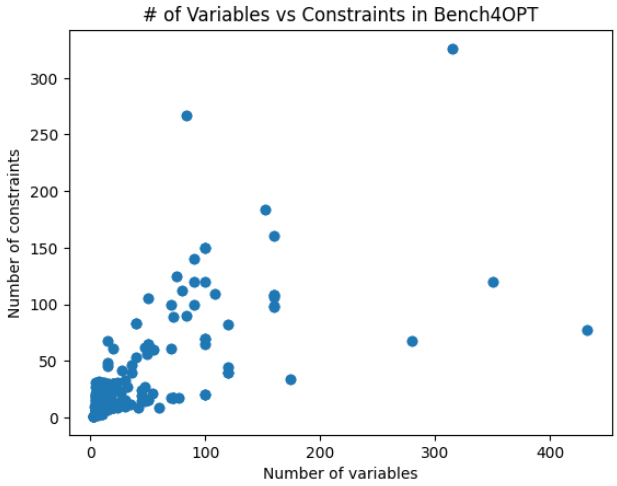

Figure 3. Optimization problem size covered in our Bench4Opt.

C. Equivalence Evaluation

C.1. Transformation of a Model Instance into a Bipartite Graph

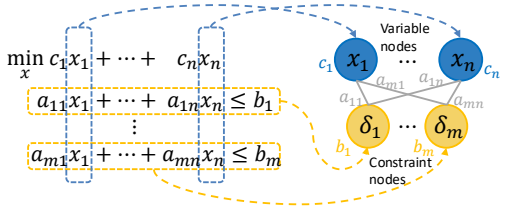

Figure 8. Transformation of a model instance into a bipartite graph.

C.2. Model Equivalence Class

Definition C.1 (Model Equivalence)**.** We say $\mathcal{C}(\mathcal{P})$ is a **model equivalence class** of the MILP/LP problem instance $\mathcal{P}$ if $\forall \hat{\mathcal{P}} \in \mathcal{C}(\mathcal{P}), \exists$ permutation matrices P_1, P_2 which shuffles the index of a vector or column index of a matrix s.t. $\hat{\mathcal{P}}$ can be written in the following form:

$$\min_x \hat{c}^T x,$$
$$\text{s.t. } \hat{A} x \hat{\circ} \hat{b},$$

where $\hat{b} = P_2 b, \hat{c} = P_1 c, \hat{A} = P_2 A P_1^\top, \hat{\circ} = P_2 \circ .$.

$\forall \mathcal{P}_2 \in \mathcal{C}(\mathcal{P}_1)$, we say $\mathcal{P}_2$ is **model-equivalent** to $\mathcal{P}_1$, denote as $\mathcal{P}_1 \sim \mathcal{P}_2$.

C.3. Weighted Bipartite Graph for Representing MILP/LP

A weighted bipartite graph for a MILP/LP instance is denoted by $\mathbf{G} = (\mathbf{V} \cup \mathbf{W}, \mathbf{E})$, with vertex set $\mathbf{V} \cup \mathbf{W}$ divided into 2 groups $\mathbf{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_m\}$ for constraints, and $\mathbf{W} = \{\mathbf{w}_1, \cdots, \mathbf{w}_n\}$ for variables, $\mathbf{E}$ consisting of $E_{ij} = E(v_i, w_j), \forall i = 1, \cdots, m, j = 1, \cdots, n$. To fully represent all information in a MILP/LP instance, we associate each vertex with features:

- The constraint vertex $\mathbf{v}_i \in \mathbf{V}$ is equipped with a feature vector $\mathbf{H}^V$ with elements $\mathbf{h}_i^V = (b_i, o_i) \in \mathcal{H}^V = \mathbb{R} \times \{\leq, \geq, =, <, >\}$
- The variable vertex $\mathbf{w}_j \in \mathbf{W}$ is equipped with a feature

```
In a cargo loading scenario, you need to choose a subset of items, each with a given value
and weight, to maximize total value without surpassing the vehicle's weight capacity. The
decision to include an item is binary. You'll be given a list of item values, weights, and
the vehicle's capacity. Your task is to determine which items to include to achieve the
highest total value while staying within the weight limit.

You should only consider parameters listed below. And these parameters will be provided in a
separated "data.json".
{
  'values': 'the value of each item; list of length (number of items)',
  'weights': 'the weight of each item; list of length (number of items)',
  'capacity': 'the capacity of the vehicle; single float value',
}
```

Figure 4. Example for concise version word problem on cargo loading.

vector $\mathbf{H}^W$ with elements $\mathbf{h}_j^W = (c_j, \tau_j) \in \mathcal{H}^W = \mathbb{R} \times \{0, 1\}$. $\tau_j = 1$ if $j \in \mathbb{Z}$ and $\tau_j = 0$ otherwise.

The edge $E_{ij} \in \mathbb{R}$ connects $\mathbf{v}_i \in \mathbf{V}$ and $\mathbf{w}_j \in \mathbf{W}$, $E_{ij} = \mathbf{A}_{ij}$. There is no edge connecting vertices in the same vertex group.

C.4. Connection between Model Equivalence and Graph Isomorphism

To test whether 2 modeling instances were permutation equivalent, we can equivalently conduct isomorphism testing between their corresponding weighted bipartite graphs. Lemma C.3 establishes an equivalence between assessing formulation correctness and graph isomorphism testing.

Definition C.2 (Graph Isomorphism)**.** Consider 2 graphs $\mathcal{G}_1 = (\mathbf{G}_1, \mathbf{H}_1^V \times \mathbf{H}_1^W)$ and $\mathcal{G}_2 = (\mathbf{G}_2, \mathbf{H}_2^V \times \mathbf{H}_2^W)$ with $\mathbf{G}_i = (\mathbf{V}^i \cup \mathbf{W}^i, \mathbf{E}^i)|_{1 \le i \le 2}$. We say $\mathcal{G}_1$ and $\mathcal{G}_2$ are **isomorphic** if there exists permutation matrix $\mathbf{P}_1, \mathbf{P}_2$ such that: $\mathbf{P}_1 \mathbf{E}^1 P_2^T = \mathbf{E}^2, \mathbf{P}_1 \mathbf{H}_1^W = \mathbf{H}_2^W, \mathbf{P}_2 \mathbf{H}_1^V = \mathbf{H}_2^V$. If 2 graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are isomorphic, denote $\mathcal{G}_1 \overset{g}{\sim} \mathcal{G}_2$.

Lemma C.3. *$\forall$ MILP/LP instances $\mathcal{P}_1, \mathcal{P}_2$ with corresponding bipartite graph $\mathcal{G}_1, \mathcal{G}_2$, we have*

$$\mathcal{P}_1 \sim \mathcal{P}_2 \Longleftrightarrow \mathcal{G}_1 \overset{g}{\sim} \mathcal{G}_2.$$

C.5. Proof of Lemma C.3

We prove this lemma by establishing two claims. We order the nodes of each bipartite graph as [variable nodes $\mathbf{W}$, constraint nodes $\mathbf{V}$], so that the adjacency matrix takes the form

$$\mathbf{A}_{\text{adj}}^{(k)} = \begin{bmatrix} 0 & \mathbf{A}_k^\top \\ \mathbf{A}_k & 0 \end{bmatrix}, \quad k = 1, 2,$$

where $\mathbf{A}_k \in \mathbb{R}^{m \times n}$ is the constraint coefficient matrix (with A_{ij} encoding the coefficient of variable j in constraint i).

Claim 1: $\mathcal{G}_1 \overset{g}{\sim} \mathcal{G}_2 \Longrightarrow \mathcal{P}_1 \sim \mathcal{P}_2$.

Suppose $\mathcal{G}_1 \overset{g}{\sim} \mathcal{G}_2$. By Definition C.2, there exist permutation matrices $\mathbf{P}_1 \in \{0, 1\}^{n \times n}$ (for variable nodes) and $\mathbf{P}_2 \in \{0, 1\}^{m \times m}$ (for constraint nodes). Define $\hat{\mathbf{P}} = \text{diag}(\mathbf{P}_1, \mathbf{P}_2)$. The graph isomorphism conditions from Definition C.2 can be equivalently written in terms of the adjacency matrix as

$$\hat{\mathbf{P}}\, \mathbf{A}_{\text{adj}}^{(1)}\, \hat{\mathbf{P}}^\top = \mathbf{A}_{\text{adj}}^{(2)}, \quad \mathbf{P}_1 \mathbf{H}_1^W = \mathbf{H}_2^W, \quad \mathbf{P}_2 \mathbf{H}_1^V = \mathbf{H}_2^V.$$

Expanding the block product yields $\mathbf{A}_2 = \mathbf{P}_2 \mathbf{A}_1 \mathbf{P}_1^\top$. The feature conditions give $\mathbf{c}_2 = \mathbf{P}_1 \mathbf{c}_1$, $\boldsymbol{\tau}_2 = \mathbf{P}_1 \boldsymbol{\tau}_1$, $\mathbf{b}_2 = \mathbf{P}_2 \mathbf{b}_1$, and $\circ_2 = \mathbf{P}_2 \circ_1$. These are precisely the conditions in the definition of formulation isomorphism, so $\mathcal{P}_1 \sim \mathcal{P}_2$.

Claim 2: $\mathcal{P}_1 \sim \mathcal{P}_2 \Longrightarrow \mathcal{G}_1 \overset{g}{\sim} \mathcal{G}_2$.

Suppose $\mathcal{P}_1 \sim \mathcal{P}_2$. By the definition of formulation isomorphism, there exist permutation matrices $\mathbf{P}_1$ $(n \times n)$ and $\mathbf{P}_2$ $(m \times m)$ such that

$$\begin{aligned} \mathbf{c}_2 &= \mathbf{P}_1 \mathbf{c}_1, \qquad \boldsymbol{\tau}_2 = \mathbf{P}_1 \boldsymbol{\tau}_1, \\ \mathbf{b}_2 &= \mathbf{P}_2 \mathbf{b}_1, \quad \mathbf{A}_2 = \mathbf{P}_2 \mathbf{A}_1 \mathbf{P}_1^\top, \\ \circ_2 &= \mathbf{P}_2 \circ_1 . \end{aligned}$$

The adjacency matrix of $\mathcal{G}_2$ is therefore

$$\begin{aligned}\mathbf{A}_{\text{adj}}^{(2)} &= \begin{bmatrix} 0 & \mathbf{A}_2^\top \\ \mathbf{A}_2 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & \mathbf{P}_1\mathbf{A}_1^\top\mathbf{P}_2^\top \\ \mathbf{P}_2\mathbf{A}_1\mathbf{P}_1^\top & 0 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{P}_1 & 0 \\ 0 & \mathbf{P}_2 \end{bmatrix} \begin{bmatrix} 0 & \mathbf{A}_1^\top \\ \mathbf{A}_1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1^\top & 0 \\ 0 & \mathbf{P}_2^\top \end{bmatrix} \\ &= \hat{\mathbf{P}}\,\mathbf{A}_{\text{adj}}^{(1)}\,\hat{\mathbf{P}}^\top.\end{aligned}$$

In addition, $\mathbf{H}_2^W = \mathbf{P}_1\mathbf{H}_1^W$ and $\mathbf{H}_2^V = \mathbf{P}_2\mathbf{H}_1^V$. By Definition C.2, $\mathcal{G}_1 \overset{g}{\sim} \mathcal{G}_2$.

```
**Problem Statement: Knapsack Problem in Cargo Loading**

**Background:**
In the context of cargo loading, the knapsack problem involves selecting a subset of items
to include in a cargo such that the total value of the selected items is maximized, while
ensuring that the total weight of the selected items does not exceed the vehicle's capacity.
This problem is a classical example of a combinatorial optimization problem and is widely
studied in operations research.

**Problem Description:**
Given a set of items, each with a specific value and weight, the objective is to determine
which items to include in the cargo to maximize the total value without exceeding the
vehicle's weight capacity. The decision to include an item in the cargo is binary (either
the item is included or it is not).

**Parameters:**
Only consider parameters listed below. And these parameters will be provided in a separated
"data.json".
{
  'values': 'the value of each item; list of length (number of items)',
  'weights': 'the weight of each item; list of length (number of items)',
  'capacity': 'the capacity of the vehicle; single float value',
}

**Decision Variables:**
- \( x[i] \): A binary variable that indicates whether item \( i \) is included in the cargo
(1) or not (0).

**Objective:**
Maximize the total value of the items included in the cargo. This is achieved by summing the
product of the value of each item and its corresponding binary decision variable.

**Constraints:**
The total weight of the items included in the cargo cannot exceed the vehicle's capacity.
This is ensured by summing the product of the weight of each item and its corresponding
binary decision variable and ensuring that this sum does not exceed the given capacity.

**Implementation Notes:**
- The problem is formulated as a Mixed-Integer Linear Programming (MILP) problem.
- The decision variables are binary, indicating the inclusion or exclusion of each item.
- The model should be saved as a '.lp' file for further analysis and solution.

**Expected Outcome:**
The expected outcome is a selection of items that maximizes the total value while ensuring
that the total weight does not exceed the vehicle's capacity. The solution will provide the
optimal set of items to include in the cargo.
```

Figure 5. Example for word problem on cargo loading.

```
import json
from gurobipy import Model, GRB

# {problem_class} — {problem_name}
# Problem type: {problem_type}
# Domain: {domain}
# Property: (fill in this comment by briefly describing the variant of the problem)

# Read data
with open('data.json', 'r') as f:
data = json.load(f)

### (fill in this section) Read parameters from data (assign domain specific parameter name)

### (fill in this section) Get hyperparameter from parameters (assign domain specific parameter name)

# Create a new model
model = Model("{problem_class}")

### (fill in this section) Add variables of the classic {problem_name} (assign domain specific name)

### (fill in this section) Set objective of the {problem_name} (assign domain specific name)

### (fill in this section) Add constraints of the {problem_name} (assign domain specific name)

# Save the model as a '.lp' file.
model.write('model.lp')
```

Figure 6. Code skeleton for optimization model simulation.

```
**Problem Statement: {problem_name} in {domain}**

**Background:**
(A brief description of background information)

**Problem Description:**
(A brief description of {problem_name} in {domain})

**Parameters:**
Only consider parameters listed below. And these parameters will be provided in a separated "data.json".
{parameter_skeleton}

**Decision Variables:**
(A list of decision variables and their description)

**Objective:**
(State the objective function)

**Constraints:**
(A list of constraints in pure natural language)

**Implementation Notes:**
(Any additional notes for implementation)

**Expected Outcome:**
(a brief description of the expected outcome)
```

Figure 7. Standard structure for word problem crafted from INFORMS AIMMS-MOPTA Optimization Modeling Competition.

C.6. Algorithms

Algorithm 2 is for conducting WL coloring for MILP/LP bipartite graphs. Algorithm 3 is for determining the symmetric decomposability of a graph.

Algorithm 2 WL test for MILP/LP Graphs

Require: A graph instance $(G, H) \in \mathcal{G}_{m,n} \times \mathcal{H}_m^V \times \mathcal{H}_n^W$ and iterate limit $L > 0$.

Initialize with $C_i^{0,V} = HASH_{0,V}(h_i^V)$, $C_j^{0,W} = HASH_{0,W}(h_j^W)$

for $l = 1, 2, \cdots, L$ **do**

$$C_i^{l,V} = HASH(C_i^{l-1,V}, \sum_{j=1}^{n} E_{i,j} HASH'_{l,W}(C_j^{l-1,W}))$$

$$C_i^{l,W} = HASH(C_i^{l-1,W}, \sum_{j=1}^{n} E_{i,j} HASH'_{l,V}(C_j^{l-1,V}))$$

return The multisets containing all colors $\{\{C_i^{L,V}\}\}_{i=0}^m, \{\{C_i^{L,W}\}\}_{j=0}^n$.

Notation: In Algorithm 3, we denote the collection of all nodes v_i's indexed by $i \in I_p$ as $\mathbf{I}_p$. Function **checkDisjointness**($[Cluster[1], \cdots, Cluster[k-1]]$) outputs "True" if any two sets $Cluster[i], Cluster[j]$, $i \neq j$, share a common element. Function **checkConnectivity**($[Cluster[1], \cdots, Cluster[k-1]]$) outputs "True" if there exist nodes $s \in Cluster[i]$ and $s' \in Cluster[j]$, $i \neq j$, such that s is connected with s'.

Algorithm 3 Determine if the graph is symmetric decomposable or not

Require: Graph $\mathcal{G}$'s adjacent matrix $\mathbf{A}$ and type 2 stable partition sets of it's variable nodes $\mathcal{I} = \{I_1, I_2, \cdots, I_{s'}\}$ and constraint nodes $\mathcal{J} = \{J_1, J_2, \cdots, J_{t'}\}$.

Ensure: Returns **True** if the graph is symmetric decomposable; otherwise, **False**.

$k \leftarrow |I_1|$.
if $|I_s| \neq k$ or $|J_t| \neq k$ for some $s = 1, \cdots, s', t = 1, \cdots, t'$. **then**
  return False
else
  Initialize an empty Cluster dictionary $Cluster$
  for $i \leftarrow 0$ to $k-1$ **do**
    $C \leftarrow$ the set of all numbers for type 2 stable partition sets
    Initialize an empty cluster set $Cluster[i]$, initialize an empty queue Q.
    while Set C is not empty **do**
      if Q is empty **then**
        Randomly select a color $c \in C$, delete c from C.
        $P_c \leftarrow$ the list of nodes labeled with $c \in C$.
        $Cluster[i] \leftarrow [P_c[i]]$, delete node $P_c[i]$ from S, push $P_c[i]$ in Q.
      else
        while Q not empty **do**
          $u \leftarrow Q.dequeue()$
          for neighborhood node w of u **do**
            if w is not in any of P_c or w is in $Cluster[i]$ **then**
              continue
            else if $color(w)$ appears in $Cluster[i]$ **then**
              return False
            else
              Add w in $Cluster[i]$, delete color(w) from C, push w in Q.
  if colors in $Cluster[i] \neq 1$ are not distinct for some $i = 0, \cdots k-1$ **then** ▷ check distinct color
    return False
  else if checkDisjointness($[S^1, \cdots, S^{k-1}]$) **then** ▷ check disjointness
    return False
  else if checkConnectivity($[S^1, \cdots, S^{k-1}]$) **then** ▷ check disconnectivity
    return False
return True

C.7. Proof Preparation for Theorem 3.3

Before establishing the proof, we first introduce the coloring refinement process of WL test for MILP/LP problem since it is the first step 1 in algorithm $\mathcal{A}$. For iteration l of the algorithm we will be assigning to each node a tuple H_i^L containing the node's old compressed label and a multiset of the node's neighbors' compressed labels. A multiset is a set (a collection of elements where order is not important) where elements may appear multiple times.

At each iteration l, we will additionally be assigning to each node a new "compressed" label C_i^L with the same H_i^L will get the same compressed label.

Repeat the above process for up to (m+n) (the number of nodes) iterations or until the partition of nodes by compressed label does not change from one iteration to the next, we will get a converged multiset.

In addition, we introduce preliminary tools for an algorithm-independent definition.

In fact, unfoldable and symmetric decomposable can be defined without relying on WL-test algorithm. We introduced equivalent definitions based on stable partition index sets.

Definition C.4 (Stable Partition Index Sets)**.** For a modeling instance $\mathcal{P}$ in the form of (1) with n decision variables and n constraints, define index set for optimization variables by $\mathcal{I} = \{I_1, I_2, \cdots, I_s\}$ and index set for constraints by $\mathcal{J} = \{J_1, J_2, \cdots, J_t\}$, where

- $\bigcup_{l=1}^{s} I_l = \{1, 2, \cdots, m\}, \bigcup_{k=1}^{t} J_k = \{1, 2, \cdots, n\}$;
- $I_{l_i} \cap I_{l_j} = \emptyset$, $J_{k_p} \cap J_{k_q} = \emptyset, \forall i, j \in [1, \cdots, |I_l|], i \neq j$, and $p, q \in [1, \cdots, |J_k|], p \neq q$.

We say $(\mathcal{I}, \mathcal{J})$ is a pair of stable partition index sets if the following condition holds:

1. $(c_i, \tau_i) = (c_{i'}, \tau_{i'}), \forall i, i' \in I_p$ for some $p \in 1, 2, \cdots, s$;
2. $(b_j, \circ_j) = (b_{j'}, \circ_{j'}), \forall j, j' \in J_q$ for some $q \in 1, 2, \cdots, t$;
3. $\forall p \in 1, 2, \cdots, s, q \in 1, 2, \cdots, t$, and $i, i' \in I_p$, we have $\sum_{j \in J_q} a_{ij} = \sum_{j \in J_q} a_{i'j}$;
4. $\forall p \in 1, 2, \cdots, s, q \in 1, 2, \cdots, t$, and $j, j' \in J_q$, we have $\sum_{i \in I_p} a_{ij} = \sum_{i \in I_p} a_{ij'}$;

Lemma C.5. *If there are no collision of hash functions and their weighted averages, then WL test algorithm 2 will finally terminated at some stable partition in $\mathcal{O}(m+n)$ iterations.*

Lemma C.5 is proved in (Chen et al., 2022b).

Definition C.6 (Unfoldable, by trivial partition)**.** $\mathcal{P}$ is unfoldable if $\exists$ stable partition index sets $\mathcal{I}$ and $\mathcal{J}$ such that $\mathcal{I}$ or $\mathcal{J}$ are trivial partitions, i.e. $s = m$ and $t = n$.

Definition C.7 (Symmetric Decomposable, by grouped partition)**.** $\mathcal{P}$ is symmetric decomposable if the following condition holds:

$\exists$ stable partition index set $\mathcal{I}$ and $\mathcal{J}$ such that:

1. There are only two types of index set in $\mathcal{I}$ and $\mathcal{J}$. Type 1 set only contains a single index. Type 2 contains several indexes, denote type 2 sets by $I_1, \cdots, I_{s'}$; $J_1, \cdots, J_{t'}$. (By WL-test coloring, nodes with index in I_i or J_j share the same color.)
2. Type 2 sets $I_1, \cdots, I_{s'}$ and $J_1, \cdots, J_{t'}$ are equal-sized with $|I_p| = |J_q| = k > 1, \forall p \in \{1, 2, \cdots, s'\}$ and $q \in \{1, 2, \cdots, t'\}$.
3. There exist k disjoint groups $S^1, \cdots, S^k$ such that $|S^i \cap I_p| = |S^i \cap I_p| = 1$; and $\forall a \in S^i, b \in S^j$ with $i \neq j$, a disconnected with b.

By Lemma C.5, we can show two sets of definitions are equivalent.

C.8. Proof of Theorem 3.3

We construct the proof by two lemmas to illustrate sufficient conditions that the result of WL test coloring can reliably infer graph isomorphism.

Lemma C.8. *Suppose $\mathcal{P}_{standard}$ is unfoldable, then $\mathcal{G}_{standard}$ and $\mathcal{G}_{test}$ shares the same coloring $\iff$ $\mathcal{G}_{standard} \sim \mathcal{G}_{test}$.*

Suppose $\mathcal{P}_{standard}$ is unfoldable, want to show $\mathcal{A}(\mathcal{G}_{test}, \mathcal{G}_{standard})$ == Equivalent $\iff$ $\mathcal{P}_{test} \sim \mathcal{P}_{standard}$.

If $\mathcal{P}_{test} \sim \mathcal{P}_{standard}$, it is trivial that $\mathcal{A}(\mathcal{G}_{test}, \mathcal{G}_{standard})$ == Equivalent.

Now, consider when $\mathcal{A}(\mathcal{G}_{test}, \mathcal{G}_{standard})$ == Equivalent and $\mathcal{P}_{standard}$ unfoldable, we have $len(\mathbb{A}_1) = len(\mathcal{C}_1)$ & $len(\mathbb{A}_2) = len(\mathcal{C}_2)$.

By the detection algorithm, every color in the multisets output by WL test must be distinct, and multisets for $\mathcal{P}_{standard}$ are the same as multisets for $\mathcal{P}_{standard}$. One stable partition of $\mathcal{G}_{standard}$ and is $\{I_1, \cdots, I_n\}, \{J_1, \cdots, J_m\}$, where I_k, J_l are single-element sets. WLOG, assume $I_k = i_k, J_l = j_l$.

Similarly, denote the stable partition of $\mathcal{G}_{test}$ by $\{I'_1, \cdots, I'_n\}, \{J'_1, \cdots, J'_m\}$, with $I'_k = [i'_k], J'_l = [j'_l]$.

Now, define a bijection mapping that shuffles $[i_1, \cdots, i_m]$ and $[j_1, \cdots, j_n]$ to get $[i'_1, \cdots, i'_m]$ and $[j'_1, \cdots, j'_n]$, denote such mapping by $\mathbf{P}$. (Since each element in

$[i_1, \cdots, i_m], [j_1, \cdots, j_n], [i'_1, \cdots, i'_m]$, or $[j'_1, \cdots, j'_n]$ is distinct, we can uniquely find such bijection).

Notice that such bijection may only map the index of $v_i^{standard}$ to the index of v_j^{test} and map the index of $w_l^{standard}$ to the index of w_p^{test}, we can separately define a bijection for decision variable index as $\mathbf{P}_1$ and a bijection for constraint index as $\mathbf{P}_2$.

Therefore, exists bijection $\mathbf{P}_1$ and $\mathbf{P}_2$ such that $\mathcal{P}_{test}$ can be written in the following form:

$$\min_x \hat{c}^T x,$$
$$\text{s.t. } \hat{A} x \hat{\circ} \hat{b}$$

where $\hat{b} = P_2 b_{standard}, \hat{C} = P_1 C_{standard}, \hat{A} = P_2 A_{standard} P_1$, $\hat{\circ} = P_2 \circ_{standard}$. This implies $\mathcal{P}_{test} \sim \mathcal{P}_{standard}$.

Lemma C.9. *Suppose $\mathcal{P}_{standard}$,$\mathcal{P}_{test}$ are decomposible symmetric, then $\mathcal{G}_{standard}$ and $\mathcal{G}_{test}$ shares the same coloring $\Longleftrightarrow$ $\mathcal{G}_{standard} \sim \mathcal{G}_{test}$.*

When $\mathcal{P}_{standard}$ is decomposible symmetric, and algorithm $\mathcal{A}$ output "Equivalent", the partition sets of $\mathcal{G}_{standard}$ and $\mathcal{G}_{test}$ can be denoted as

$$\mathcal{I}_{standard} = [I_1, \cdots, I_k, I_{k+1}, \cdots, I_s];$$
$$\mathcal{J}_{standard} = [J_1, \cdots, J_l, J_{l+1}, \cdots, J_t];$$
$$\mathcal{I}_{test} = [\hat{I}_1, \cdots, \hat{I}_k, \hat{I}_{k+1}, \cdots, \hat{I}_s];$$
$$\mathcal{J}_{test} = [\hat{J}_1, \cdots, \hat{J}_l, \hat{J}_{l+1}, \cdots, \hat{J}_t],$$

Here $[I_1, \cdots, I_k]$, $[\hat{I}_1, \cdots, \hat{I}_k]$, $[J_1, \cdots, J_l]$, and $[\hat{J}_1, \cdots, \hat{J}_l]$ only contain singleton sets. In contrast, $[I_{k+1}, \cdots, I_s]$, $[\hat{I}_{k+1}, \cdots, \hat{I}_s]$, $[J_{l+1}, \cdots, J_t]$, and $[\hat{J}_{l+1}, \cdots, \hat{J}_t]$ each contain sets with at least two indices. Moreover, I_i and $\hat{I}_i$ share the same color for all i, and J_j and $\hat{J}_j$ share the same color for all j.

Now, define a bijection mapping that maps $[I_1, \cdots, I_k, I_{k+1}, \cdots, I_s, J_1, \cdots, J_l, J_{l+1}, \cdots, J_t]$ to $[\hat{I}_1, \cdots, \hat{I}_k, \hat{I}_{k+1}, \cdots, \hat{I}_s, \hat{J}_1, \cdots, \hat{J}_l, \hat{J}_{l+1}, \cdots, \hat{J}_t]$, by the following rules:

1. For the unique index $i \in I_p$, where $p \in \{1, \cdots, k\}$, map i to the unique index $i' \in \hat{I}_p$.
2. For the unique index $j \in J_q$, where $q \in \{1, \cdots, l\}$, map j to to the unique index $j' \in \hat{J}_q$.
3. For the remaining nodes, we consider a cluster-wise mapping, i.e finding some equivalent clusters, mapping a cluster to another, and providing a unique mapping rule within choosen cluster.

   Let V' and $\hat{V}'$ be the sets of all variable nodes except those with unique colors in $\mathcal{G}_{standard}$ and $\mathcal{G}_{test}$, respectively. Let W' and $\hat{W}'$ be the sets of all constraint nodes except those with unique colors in $\mathcal{G}_{standard}$ and $\mathcal{G}_{test}$, respectively. Find clusters $S^1, \cdots, S^r$ such that each S^i is disconnected, disjoint, and consists of nodes with the same combination of unique colors as the other clusters, with $\bigcup_{i=1}^r S^i = V' \cup W'$. Similarly, for symmetric decomposable $\mathcal{G}_{test}$ with the same coloring distribution, we can find clusters $\hat{S}^1, \cdots, \hat{S}^r$ such that $\hat{S}^i$ has the same coloring distribution as S^i, each $\hat{S}^i$ is disconnected and disjoint, and $\bigcup_{i=1}^r \hat{S}^i = \hat{V}' \cup \hat{W}'$.

   The existence of $S^1, \cdots, S^r$ and $\hat{S}^1, \cdots, \hat{S}^r$ are guaranteed by the symmetric decomposable property of $\mathcal{G}_{standard}$ and $\mathcal{G}_{test}$.

   Now, we can define a bijection that maps S^i to a corresponding cluster $\hat{S}_i$. Note that nodes in one cluster have distinct colors. The bijection mapping maps nodes from cluster S^i to $\hat{S}_i$ according to color-matching, i.e. a node maps to another one when they are in the same color.

Now, consider the adjacency matrix of the representing bipartite graph

$$\mathbf{A}_{adj} = \begin{bmatrix} 0 & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}.$$

Since node groups $S^1, \cdots, S^k$ are disconnected, we can rearrange matrix A by some column permutation $\mathbf{P}_1^b$ and row permutation $\mathbf{P}_2^b$ such that

$$\mathbf{P}_1^b \mathbf{A} \mathbf{P}_2^b = \begin{bmatrix} \mathbf{A}_1 & 0 & \cdots & 0 & \mathbf{a_1} \\ 0 & \mathbf{A}_2 & \cdots & 0 & \mathbf{a_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_r & \mathbf{a_r} \\ \mathbf{b_1}^T & \mathbf{b_2}^T & \cdots & \mathbf{b_r}^T & \mathbf{A}_{r+1} \end{bmatrix}$$

where $A_1, \cdots, A_r$ are coefficient matrix for r clusters $S^1, \cdots, S^r$ with associated decision variables and constraints, and A_{r+1} is a $k \times l$ matrix.

The above composition of bijection mapping operations is equivalent to applying permutation operations on $A_{standard}, b_{standard}, c_{standard}, \circ_{standard}$ by the following steps:

1. point-wise mapping for variables: permute c and A by permutation matrix $\mathbf{P}_1^0$ to map unique index $i \in I_p$ to $i' \in \hat{I}_p$, which produce $\hat{c} = \mathbf{P}_1^0 c_{standard}$ and $\hat{A} = A_{standard} \mathbf{P}_1^0$
2. point-wise mapping for constraints: permute $b_{standard}, \circ_{standard}$ and $\hat{A}$ by permutation matrix

$\mathbf{P}_2^0$ to map unique index $j \in I_q$ to $j' \in \hat{J}_q$, which produce $\hat{b} = \mathbf{P}_2^0 b_{standard}$, $\hat{\circ} = \mathbf{P}_2^0 \circ_{standard}$ and $\hat{A} = \mathbf{P}_2^0 \hat{A} = \mathbf{P}_2^0 A_{standard} \mathbf{P}_1^0$

3. clustering mapping: permute $\hat{c}$ and $\hat{A}$ by permutation matrix $\mathbf{P}_1^c$ and permute $\hat{b}, \hat{\circ}$ and $\hat{A}$ by permutation matrix $\mathbf{P}_2^c$ to produce

$$\hat{A} = \mathbf{P}_1^c \hat{A} \mathbf{P}_2^c = \begin{bmatrix} \mathbf{A}_1 & 0 & \cdots & 0 & \mathbf{a_1} \\ 0 & \mathbf{A}_2 & \cdots & 0 & \mathbf{a_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_r & \mathbf{a_r} \\ \mathbf{b_1}^T & \mathbf{b_2}^T & \cdots & \mathbf{b_r}^T & \mathbf{A}_{r+1} \end{bmatrix}$$
$$= \mathbf{P}_1^c \mathbf{P}_1^0 A_{standard} \mathbf{P}_2^0 \mathbf{P}_2^c,$$

$\hat{b} = \mathbf{P}_2^c \mathbf{P}_2^0 b_{standard}$, $\hat{\circ} = \mathbf{P}_2^c \mathbf{P}_2^0 \circ_{standard}$, and $\hat{c} = \mathbf{P}_1^c \mathbf{P}_1^0 c_{standard}$

4. in-cluster mapping: iteratively permute $\hat{c}$ and $\hat{A}$ by permutation matrices $\mathbf{P}_1^1, \cdots, \mathbf{P}_1^r$ and permute $\hat{b}, \hat{\circ}$ and $\hat{A}$ by permutation matrices $\mathbf{P}_2^1, \cdots, \mathbf{P}_2^r$ to produce $\hat{A} = \mathbf{P}_1^r \cdots \mathbf{P}_1^1 \mathbf{P}_1^c \mathbf{P}_1^0 A_{standard} \mathbf{P}_2^0 \mathbf{P}_2^c \mathbf{P}_2^1 \cdots \mathbf{P}_2^r$, $\hat{b} = \mathbf{P}_2^k \cdots \mathbf{P}_2^1 \mathbf{P}_2^b \mathbf{P}_2^0 b_{standard}$, $\hat{\circ} = \mathbf{P}_2^r \cdots \mathbf{P}_2^1 \mathbf{P}_2^b \mathbf{P}_2^0 \circ_{standard}$, and $\hat{c} = \mathbf{P}_1^r \cdots \mathbf{P}_1^1 \mathbf{P}_1^c \mathbf{P}_1^0 c_{standard}$

Now, define $\mathbf{P}_1 = \mathbf{P}_1^k \cdots \mathbf{P}_1^1 \mathbf{P}_1^c \mathbf{P}_1^0$ and $\mathbf{P}_2 = \mathbf{P}_2^k \cdots \mathbf{P}_2^1 \mathbf{P}_2^c \mathbf{P}_2^0$, we can write $\mathcal{P}_{test}$ in the following form:

$$\min_x \hat{c}^T x,$$
$$\text{s.t. } \hat{A} x \hat{\circ} \hat{b}$$

where $\hat{b} = P_2 b_{standard}, \hat{C} = P_1 C_{standard}, \hat{A} = P_2 A_{standard} P_1$, $\hat{\circ} = P_2 \circ_{standard}$. This implies $\mathcal{P}_{test} \sim \mathcal{P}_{standard}$.

C.9. Complexity Analysis

For the two main types of problem realizations in our benchmark, Algorithm 2 converges in $\mathcal{O}(m+n)$ iterations. In addition, for problems with m variables and n constraints, the time complexity to distinguish tested problem realizations from the standard realization is at most $\mathcal{O}(kmn)$, which is significantly lower than classical algorithms employed by popular solvers, such as the simplex method for LP and the branch and bound algorithm for MILP. Specifically,

1. **For unfoldable problem instances**, algorithm 2 converges in at most $\mathcal{O}(m+n)$ iterations according to lemma C.5.

2. **For symmetric decomposable problem instances**, algorithm 2 converges in at most $\mathcal{O}(m+n)$ iterations, and we shall further conduct symmetric decomposable detection using algorithm 3, which takes time complexity $\mathcal{O}(kmn)$ in the worst case, where k is the number of clusters in the symmetric decomposable graph. The total time complexity could be $\mathcal{O}(kmn)$.

C.10. Randomly sampling suffices to obtain symmetric decomposable

To make WL test work, it is desirable to sample a symmetric decomposable instance. In Theorem C.11 and C.12, we proved that for a large range of modeling problems with reasonable assumptions, we can sample a symmetric decomposable instance from its **parameter support** with probability 1.

Definition C.10 (Modeling Parameter Support)**.** For a class of model formulation $\mathcal{M}$ with n decision variables and m constraints, the **parameter set** $\Theta(\mathcal{M})$ is a collection of all possible values for instance parameter $(\mathbf{A}, \mathbf{c}, \mathbf{b}, \circ, \tau)$. The parameter set associated with decision variable x_i is $\Theta(\mathcal{M}, i) = \{[\mathbf{A}_{:,i}^T, c_i]\}$.

An example of a formulation parameter support is attached in Appendix D.

Theorem C.11 (Efficient Sampling - continuous case)**.** *Suppose a model $\mathcal{M}$ satisfies the following conditions:*

For each $\vec{\theta}_i \in \mathbb{R}^d, i = 1, \cdots, n$, there exists a coordinate k_i such that $\vec{e}_{k_i}^\top \vec{\theta}_i$ follows a continuous distribution μ_i independently across i.

*then a random draw $\vec{\theta} \sim \Theta$ yields a **symmetric decomposable** instance $\mathcal{M}(\vec{\theta})$ almost surely.*

Theorem C.12 (Efficient Sampling - discrete case)**.** *Suppose a model $\mathcal{M}$ satisfies the following conditions:*

$\forall i = 1, \cdots, n, \forall \vec{\theta}_i \in \mathbf{R}^d, \exists k_i \in \{1, \cdots, d\}$ such that $\vec{e}_{k_i}^\top \vec{\theta}_i \sim \mu_i(\cdot)$ and independent of the distribution of $\vec{\theta}_j, \forall j \neq i$, where $\vec{e}_{k_i}$ is the k_i-th standard basis vector in $\mathbf{R}^d$, $\mu_i(\cdot)$ is some discrete uniform distribution with $u_i(\vec{e}_{k_i}^\top \vec{\theta}_i) \sim Uniform\{x_1 \cdots x_l\}$, i.e. at lease one coordinate of $\vec{\theta}_i$ can be randomly sampled with probability $\frac{1}{l}$, where k_i is the index of coordinate in $\vec{\theta}_i$ that being sampled.

Then, as $l \to \infty$, randomly sample $\vec{\theta}$ from parameter support Θ, we can get a symmetric decomposable instance for model $\mathcal{M}$ with probability 1.

We present the proof for Theorem C.11 and C.12 in Appendix C.11.

C.11. Proof of Theorem C.11 and Theorem C.12

Proof:

Lemma C.13. *Suppose model $\mathcal{M}$ satisfies the following assumption:*

$\forall i = 1, \cdots, n. \forall \vec{\theta}_i \in \mathcal{R}^d, \exists k_i \in \{1, \cdots, d\}$ *such that* $\vec{e}_{k_i}^\top \vec{\theta}_i \sim \mu_i\left(\vec{\theta}_i\right)$ *and independent of the distribution of* $\vec{\theta}_j$*, where* $\vec{e}_{k_i}$ *is the* k_i*-th standard basis vector in* $\mathbf{R}^d$*,* $\mu_i(\vec{\theta}_i)$ *is some continuous distribution; i.e., at least one coordinate of* $\vec{\theta}_i$ *can be randomly sampled according to some continuous distribution.*

Then, we have $P(\vec{\theta}_i = \vec{\theta}_j) = 0, \forall i \neq j$.

Proof of lemma C.13:

Consider $i \neq j$, $0 \leq P\left(\vec{\theta}_i = \vec{\theta}_j\right) \leq P\left(\vec{e}_{k_j}\vec{\theta}_j = \vec{e}_{k_j}\vec{\theta}_i\right) = 0$ since μ_i is continuous distribution.

By lemma C.13, we have $P\left(\vec{\theta}_i \neq \vec{\theta}_j\right) = 1$.

Lemma C.14. *Suppose model* $\mathcal{M}$ *satisfies the following condition:*

$\forall i = 1, \cdots, n, \forall \vec{\theta}_i \in \mathbf{R}^d, \exists k_i \in \{1, \cdots, d\}$ *such that* $\vec{e}_{k_i}^\top \vec{\theta}_i \sim \mu_i(\cdot)$ *and independent of the distribution of* $\vec{\theta}_j, \forall j \neq i$*, where* $\vec{e}_{k_i}$ *is the* k_i*-th standard basis vector in* $\mathbf{R}^d$*,* $\mu_i(\cdot)$ *is some discrete uniform distribution with* $u_i(\vec{e}_{k_i}^\top \vec{\theta}_i) \sim Uniform\{x_1 \cdots x_l\}$*, i.e. at least one coordinate of* $\vec{\theta}_i$ *can be randomly sampled with probability* $\frac{1}{l}$*, where* k_i *is the index of coordinate in* $\vec{\theta}_i$ *that being sampled.*

Then $P(\vec{\theta}_i = \vec{\theta}_i) \to 0$ *as* $l \to \infty$.

Proof of lemma C.14:

$$\begin{aligned} P\left(\vec{\theta}_i = \vec{\theta}_j\right) &= P\left(\vec{e}_{k_i}\vec{\theta}_i = \vec{e}_{k_i}\vec{\theta}_j\right) \\ &= \sum_x P\left(\vec{e}_{k_i}\vec{\theta}_j = x \mid \vec{e}_{k_i}\vec{\theta}_i = x\right) \\ &\quad \cdot P\left(\vec{e}_{k_i}\vec{\theta}_i = x\right) \\ &= \sum_x P\left(\vec{e}_{k_i}\vec{\theta}_j = x\right) \\ &= \sum_x \frac{1}{l^2} \\ &= \frac{1}{l}. \end{aligned}$$

as $l \to \infty$, $P\left(\vec{\theta}_i = \vec{\theta}_j\right) \to 0$.

Lemma C.15. *Suppose a modeling instance* $\mathcal{P}$ *has* $P(\vec{\theta}_j = \vec{\theta}_{j'}) = 0, \forall j \neq j'$*, then* $P(\mathcal{P}$ *is symmetric decomposable*$)$ *= 1.*

$P(\vec{\theta}_j \neq \vec{\theta}_{j'}) = 1,$ for all $j \neq j'$

Proof of lemma C.15:

Suppose $\exists$ index set $k \subset \{1, 2, \cdots, d\}$ such that $\forall k \in k, \vec{e}_k\vec{\theta}_j \neq \vec{e}_k\vec{\theta}_j$, want to show the joint probability of the following event is 1 :

1. Event A: $c_j \neq c_{j'}$. [Objective coefficients are not the same.]
2. Event B: $\sum_{i \in I} a_{ij} \neq \sum_{i \in I} a_{ij'}$ [accumulated edge weights for variable nodes of j, j' are not the same].
3. Event C: $\sum_{q \in J} a_{i'q} \neq \sum_{q \in J} a_{iq}$ for some J containing index j or j' and some $i \neq i' \in I$. [accumulated edge weights for two constraint nodes are not the same];

where I and J are sets in stable partitions $\mathcal{I}, \mathcal{J}$. It is euqivalent to show $P(A \cup B \cup C) = 1$. Now, consider two cases when $j, j' \in \{1, \cdots, n\}$:

1. Case 1: $\exists k \in K$ s.t. $\vec{e}_k^\top \vec{\theta}_j = c_j$, $\vec{e}_k^\top \vec{\theta}_{j'} = c_{j'}$, then $c_j \neq c_{j'}$.
2. Case 2: $\exists k \in K$ s.t. $\vec{e}_k^\top \vec{\theta}_j = a_{ij}$,$\vec{e}_k^\top \vec{\theta}_{j'} = a_{ij'}$ for some i, then $a_{ij} \neq a_{ij'}$ for some i.

Notice that $P(\Omega) = P(\text{Case 1} \cup \text{Case 2}) = 1$.

It suffices to show $P(A \cup B \cup C \mid \text{Case 1} \cup \text{Case 2}) = 1$. Now, $P(A \cup B \cup C \mid \text{Case 1}) = 1$ since $P(A \mid \text{Case 1}) = 1$. It suffices to show $P(A \cup B \cup C \mid \text{Case 2}) = 1$. It suffices to show $P(B \cup C \mid \text{Case 2}) = 1$.

Now, suppose $\exists k \in K$ such that $\vec{e}_k^\top \vec{\theta}_j = a_{ij} \neq \vec{e}_k^\top \vec{\theta}_{j'} = a_{ij'}$.

Consider I containing i. WLOG, suppose $\hat{I} \subset I$ is an index set that containing all $i's$ such that $a_{ij} \neq a_{ij'}$, and $\sum_{i \in I/\hat{I}} (a_{ij} - a_{ij'}) = c$ for come constant c, then

$$\begin{aligned} P\left(\sum_{i \in I} a_{ij} \neq \sum_{i \in I} a_{ij'}\right) &= P\left(\sum_{i \in \hat{I}} a_{ij} \neq \sum_{i \in \hat{I}} a_{ij'} + c\right) \\ &= 1 - P\Big(\sum_{i \in \hat{I}} a_{ij} \\ &\qquad = \sum_{i \in \hat{I}} a_{ij'} + c\Big) \\ &= 1 - 0 \\ &= 1 \end{aligned}$$

The third equality holds since $\sum_{i \in \hat{I}} a_{ij}$ and $\sum_{i \in \hat{I}} a_{ij'}$ are independent and can be sampled from some continuous

distribution. Therefore, since $P(B \mid \text{case 2}) = 0$, we have

$$\begin{aligned} P(A \cup B \cup C) &= P(A \cup B \cup C \mid \Omega) \\ &= P(A \cup B \cup C \mid \text{case 1} \cup \text{case 2}) \\ &= 1. \end{aligned}$$

Now, by lemma C.13 and lemma C.15, we can prove Theorem C.11; by lemma C.14 and lemma C.15, we can prove Theorem C.12.

D. Examples

D.1. Examples for limitations of solver-based evaluation

Example D.1 (The solver returns values and solver-based evaluation judges the model as correct, but the mathematical model is actually wrong). Consider a car production and revenue maximization problem. A manufacturer produces two types of cars: sedans and SUVs. Let the decision variables of x be the number of sedans to produce and y be the number of SUVs to produce. The correct formulation is:

$$\begin{aligned} &\text{Maximize } 30x + 50y \\ &\text{such that: } x + 2y \leq 100 \quad \text{(Labor capacity)} \\ &\qquad\quad x \geq 0, y \geq 0 \quad \text{(Non-negativity)} \end{aligned}$$

Now suppose an LLM generates an incorrect model with an additional erroneous constraint:

$$\begin{aligned} &\text{Maximize } 30x + 50y \\ &\text{such that: } x + 2y \leq 100 \quad \text{(Labor capacity)} \\ &\qquad\quad x + y \leq 40 \quad \text{(ERRONEOUS constraint)} \\ &\qquad\quad x \geq 0, y \geq 0 \quad \text{(Non-negativity)} \end{aligned}$$

If we test with a data configuration θ where production capacity = 80 and the market demand limit is 40, both models will yield the same optimal solution and optimal value: produce 40 SUVs for a revenue of $2,000. However, if the data configuration changes to θ' with production capacity = 200, the correct model would recommend producing 100 SUVs for a revenue of $5,000, while the incorrect model would still limit production to 40 units total due to the erroneous constraint.

Example D.2 (The solver returns constant value, and its useless for modeling equivalence detection). Consider a facility location problem where the goal is to determine whether it is possible to open a subset of facilities to serve all customer demand within a fixed budget. The objective is a constant (e.g., 0), since only feasibility is of interest:

$$\begin{aligned} &\text{Minimize } 0 \\ &\text{such that: } \sum_{j \in \mathcal{F}} x_j \cdot c_j \leq B \quad \text{(Budget)} \\ &\qquad\quad \sum_{j \in \mathcal{F}} a_{ij} x_j \geq 1 \quad \forall i \in \mathcal{D} \quad \text{(Coverage)} \\ &\qquad\quad x_j \in \{0, 1\} \quad \forall j \in \mathcal{F} \end{aligned}$$

Now, suppose the LLM generates an incorrect model with slightly relaxed constraints:

$$\begin{aligned} &\text{Minimize } 0 \\ &\text{such that: } \sum_{j \in \mathcal{F}} x_j \cdot c_j \leq B \quad \text{(Budget)} \\ &\qquad\quad \sum_{j \in \mathcal{F}} a_{ij} x_j \geq 0.5 \quad \forall i \in \mathcal{D} \quad \text{(ERRONEOUS)} \\ &\qquad\quad x_j \in \{0, 1\} \quad \forall j \in \mathcal{F} \end{aligned}$$

If both models happen to be feasible under a specific data configuration θ (e.g., a small number of facilities with low costs and high coverage), then the solver will return "feasible" for both. However, the second model allows partial coverage (due to the threshold of 0.5), which violates the intended semantics. Since the objective function is constant, solver-based evaluation cannot detect this structural mistake.

Example D.3 (The mathematical model is incorrect but solver-based evaluation is uninformative for infeasible problems). Consider the same car production problem, but with modified constraints:

$$\begin{aligned} &\text{Maximize } 30x + 50y \\ &\text{such that: } x + 2y \leq 10 \quad \text{(Labor capacity)} \\ &\qquad\quad x \geq 20, y \geq 0 \quad \text{(Minimum sedans)} \end{aligned}$$

This correct model is genuinely infeasible because the minimum sedan production requirement ($x \geq 20$) cannot be satisfied with the limited production capacity ($x + 2y \leq 10$). Now suppose an LLM generates an incorrect model with an erroneous constraint:

$$\begin{aligned} &\text{Maximize } 30x + 50y \\ &\text{such that: } x + 2y \leq 10 \quad \text{(Labor capacity)} \\ &\qquad\quad x \geq 5, y \geq 7 \quad \text{(ERRONEOUS minimums)} \\ &\qquad\quad x, y \geq 0 \quad \text{(Non-negativity)} \end{aligned}$$

For the data configuration θ shown above, both models will be evaluated as infeasible by the solver. The solver-based evaluation metric cannot distinguish between the correct

model that is genuinely infeasible under this configuration and the incorrect model that is infeasible due to contradictory constraints ($x \geq 5$ and $y \geq 7$ would require at least 19 labor-hours, exceeding the 10 available).

D.2. Examples for model, model parameter set, and model instance

Example D.4 (Problem Data Support). Consider the following problem: A farmer who seeks to maximize profit using t acres of land to grow wheat and rice. Wheat yields m_1 kilograms per acre and rice yields m_2 kilograms per acre; each kilogram of wheat brings a profit of c_1, and each kilogram of rice brings a profit of c_2. Then the problem data support is:

$$\Theta = T \times M_1 \times M_2 \times C_1 \times C_2 \subseteq \mathbb{R}^5_+,$$

containing all possible problem data (t, m_1, m_2, c_1, c_2).

A ground-truth model is:

$$\begin{aligned}\mathcal{M} : (t, m_1, m_2, c_1, c_2) \mapsto \min_{x_1, x_2} \; & -m_1 c_1 x_1 - m_2 c_2 x_2 \\ \text{s.t. } & x_1 + x_2 \leq t, \\ & x_1 \geq 0, \\ & x_2 \geq 0.\end{aligned}$$

Example D.5 (Model Parameter Set for Blending Problem). For example, a blending problem can be formulated as:

$$\begin{aligned}\min_x \; & \sum_{i=1}^n c_i x_i \\ \text{s.t. } & \sum_{i=1}^n a_{ji} x_i \geq p_j, \forall j = 1, \cdots, m. \\ & x_i \leq u_i, \forall i = 1, \cdots, n.\end{aligned}$$

The corresponding parameter set $\Theta(\mathcal{M}_{blend})$ can be defined by

$$\begin{aligned}\mathbf{A} &= [\hat{\mathbf{A}}^T, I_n]^T, & \hat{\mathbf{A}} &\in \mathbb{R}^{m \times n}, \\ \mathbf{c} &= [c_1, \cdots, c_n]^T, & \mathbf{c} &\in \mathbb{R}^n, \\ \mathbf{b} &= [-p_1, \cdots, -p_m, -u_1, \cdots, -u_n]^T, & \mathbf{b} &\in \mathbb{R}^{m+n}, \\ \circ &= [\geq, \cdots, \geq, \leq, \cdots, \leq]^T.\end{aligned}$$

Here I_n is the $n \times n$ identity matrix, so

$$\Theta(\mathcal{M}_{blend}) = \{(\mathbf{A}, \mathbf{c}, \mathbf{b}, \circ) \mid \text{the relations above hold}\}.$$

The parameter set associated with x_i is $\Theta(\mathcal{M}_{blend}, i) = \left\{[\mathbf{A}^T_{:,i}, c_i]\right\} = \mathbb{R}^{m+1}$.

D.3. Examples for symmetry

Example D.6 (Undesirable Symmetry). Discriminating problem instances involving symmetry in their decision variables or constraints can be tricky, because some non-isomorphic bipartite graphs cannot be distinguished by WL-test due to their automorphic structure in the graph. For example, (Chen et al., 2022b) illustrates one case in which two MILP graphs are non-isomorphic while WL-test outputs the same multiset.

Example D.7 (Symmetric Decomposable Problem: Bin–Packing). For symmetric decomposable problems, their corresponding bipartite graph can be divided into several symmetric sub-graphs, each isomorphic and disconnected from the others. For example, an instance of bin-packing with heterogeneous vehicles is formulated as

$$\begin{aligned}\min_{x \in \{0,1\}^q, y \in \{0,1\}^p} \; & \sum_{j=1}^p y_j \\ \text{s.t. } & \sum_i s_i x_{ij} \leq b y_j, \forall j = 1, \cdots, p. \\ & \sum_{j=1}^p x_{ij} = 1, \forall i = 1, \cdots, q\end{aligned}$$

For the bin-packing problem with $p = 3$ and $q = 2$, a corresponding bipartite is illustrated in figure 10, where the red node represents decision variables and the blue nodes represent constraints.

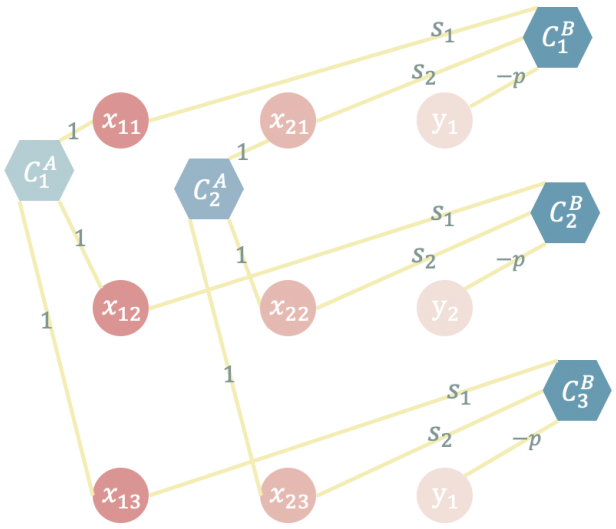

Figure 10. Bipartite for a bin-packing problem. Different colors indicate that the nodes are colored using the WL test. This figure illustrates the representation of a symmetric decomposable graph. There are four groups of nodes with the same colors in each group, and two nodes with distinct colors. In addition, a node in any group, for example, the lightest red group, only connects with one node in other groups.

Such graphs are quite special since by excluding uniquely colored nodes and their connecting edges, the remaining

$$
\begin{aligned}
\min_{x\in\mathbb{R}^6}\ & x_1+x_2+x_3+x_4+x_5+x_6,\\
\text{s.t. }& x_1+x_2=1,\ x_2+x_3=1,\ x_3+x_4=1,\\
& x_4+x_5=1,\ x_5+x_6=1,\ x_6+x_1=1,\\
& 0\le x_j\le 1,\ x_j\in\mathbb{Z},\ \forall\, j\in\{1,2,\dots,6\}.
\end{aligned}
$$

$$
\begin{aligned}
\min_{x\in\mathbb{R}^6}\ & x_1+x_2+x_3+x_4+x_5+x_6,\\
\text{s.t. }& x_1+x_2=1,\ x_2+x_3=1,\ x_3+x_1=1,\\
& x_4+x_5=1,\ x_5+x_6=1,\ x_6+x_4=1,\\
& 0\le x_j\le 1,\ x_j\in\mathbb{Z},\ \forall\, j\in\{1,2,\dots,6\}.
\end{aligned}
$$

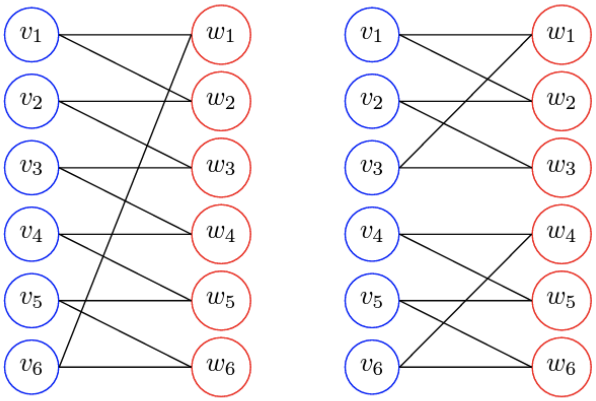

Figure 9. Two non-isomorphic MILP graphs that cannot be distinguished by the WL test

symmetric nodes (nodes labeled in the same color via the WL test) can be combined to form several isomorphic, disconnected, and unfoldable graphs, as the dashed line highlights in Figure 11.

Example D.8 (Symmetric Decomposable Problem: Set-Covering). Consider a public transportation planning task in which a city aims to select a minimal set of bus stops such that all residential areas are served. Let S denote the set of potential stop locations and A the set of residential areas. The service relationship is encoded in a binary coverage matrix C, where $C_{ij}=1$ if stop $j\in S$ can serve residential area $i\in A$, and $C_{ij}=0$ otherwise. The objective is to choose the smallest subset of stops that collectively cover all residential areas.

To model this decision problem, we define a binary decision variable y_j for each potential stop $j\in S$, where $y_j=1$ indicates that stop j is selected. The resulting optimization model is:

$$
\begin{aligned}
\min_{y_j}\quad & \sum_{j\in S} y_j\\
\text{s.t.}\quad & \sum_{j\in S} C_{ij}\,y_j\ge 1, && \forall i\in A,\\
& y_j\in\{0,1\}, && \forall j\in S.
\end{aligned}
$$

Now, consider a small example with 4 potential stops and 4 residential areas. The set of potential stops is

$$S=\{0,1,2,3\},$$

and the set of residential areas is

$$A=\{0,1,2,3\}.$$

The coverage relationship is encoded in the binary matrix C, where $C_{ij}=1$ indicates that stop j covers area i. For

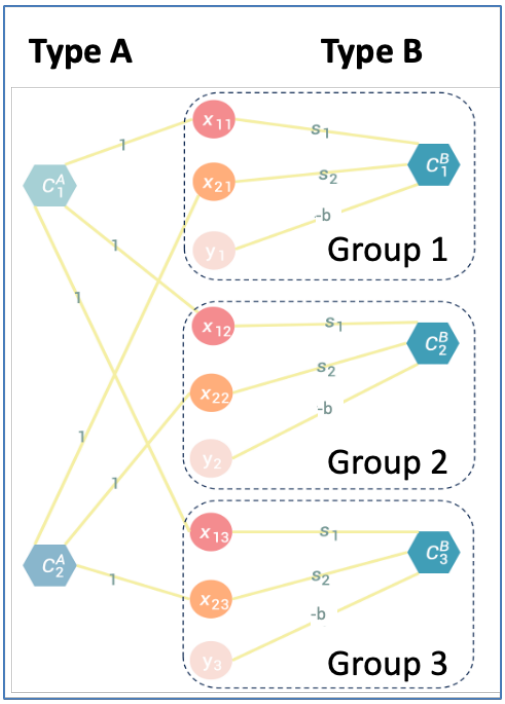

Figure 11. Decompose a symmetric decomposable graph for a bin-packing instance: After applying the WL test, the graph contains two types of nodes: **Type-A**: uniquely colored nodes (C_1^A, C_2^A); **Type-B**: 3 groups of nodes that share the same color pattern. Each group contains nodes with distinct colors, the two groups have identical color distributions, and there are no edges between the groups.

this instance, the coverage matrix is

$$C = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

Each row corresponds to a residential area, and each column corresponds to a potential stop. For example, the first row indicates that area 0 is covered by stops 2 and 3, and the second row shows that area 1 is covered only by stop 3.

Substituting these parameters into the general set covering formulation yields the following optimization problem:

$$\begin{aligned} \min_{y_j \in \{0,1\}} \quad & y_0 + y_1 + y_2 + y_3 \\ \text{s.t.} \quad & y_2 + y_3 \geq 1, \\ & y_3 \geq 1, \\ & y_0 + y_2 \geq 1, \\ & y_0 + y_1 \geq 1. \end{aligned}$$

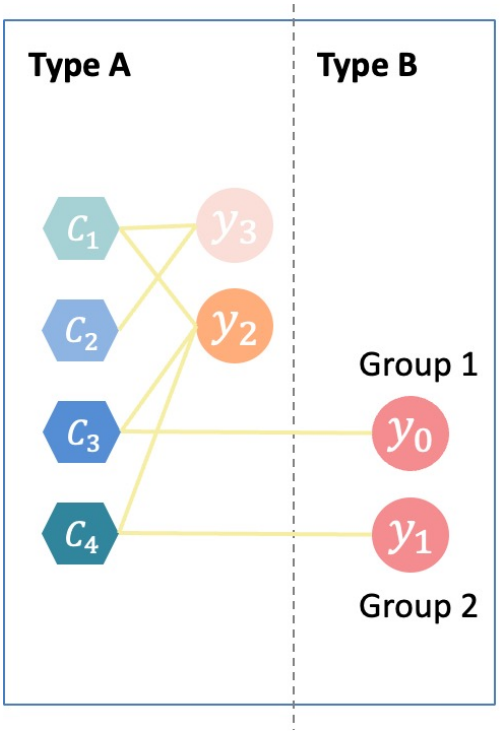

Figure 12. Decompose a symmetric graph for a set-covering instance: After applying the WL test, the graph contains two types of nodes: **Type-A**: uniquely colored nodes ($C_1, C_2, C_3, C_4, y_2, y_3$); **Type-B**: two groups of nodes that share the same color pattern. Each group contains nodes with distinct colors, the two groups have identical color distributions, and there are no edges between the groups.

As illustrated by the corresponding bipartite graph in Figure 12, the set-covering instance is symmetrically decomposable.

Example D.9 (Symmetric Decomposable Problem Instance: Capital Budgeting Problem). Consider a capital budgeting problem in which a firm must decide which projects to invest in over multiple planning stages. Each project provides a Net Present Value (NPV) and requires a certain investment cost as well as multiple types of resources. Denote the set of candidate projects and T the set of planning stages. The value of project $i \in I$ is given by NPV_i, and its investment cost is denoted by c_i. Projects also consume resources: project i uses a_{ir} units of resource type r out of a per-stage resource availability R_{tr}.

To model sequential investment decisions, we introduce binary decision variables x_{it} indicating whether project i is undertaken in stage t. The goal is to maximize the total NPV accumulated through investments across all stages. The optimization model is:

$$\begin{aligned} \max_{x_{it}} \quad & \sum_{t \in T} \sum_{i \in I} \mathrm{NPV}_i \, x_{it} \\ \text{s.t.} \quad & \sum_{i \in I} a_{ir} \, x_{it} \leq R_{tr}, \quad \forall t \in T, \ \forall r, \\ & x_{it} \in \{0,1\}, \qquad \forall i \in I, \ \forall t \in T. \end{aligned}$$

Now, let

- $I = \{1, 2, 3\}$ be the index set of candidate projects.
- $T = \{1, 2\}$ be the set of planning stages.
- NPV_i denotes the net present value of project i. The objective coefficient for project i in every stage is NPV_i.
- r_{ir} denotes the consumption of resource type r by project i. Here $r = 1, 2$ indexes two distinct resource types.
- R_{tr} denotes the availability (limit) of resource r at stage t.
- $x_{it} \in \{0, 1\}$ is a binary decision variable equal to one if project i is undertaken in stage t, and zero otherwise.

Then we have an instance for capital budgeting problem:

$$\begin{aligned} \max \quad & \sum_{t=1}^{2} \sum_{i=1}^{3} \mathrm{NPV}_i \, x_{it} \\ \text{s.t.} \quad & \sum_{i=1}^{3} r_{ir} \, x_{it} \leq R_{tr}, \\ & \forall t, r \in \{1, 2\}, \\ & x_{it} \in \{0, 1\}, \\ & \forall i \in \{1, 2, 3\}, \ \forall t \in \{1, 2\}. \end{aligned}$$

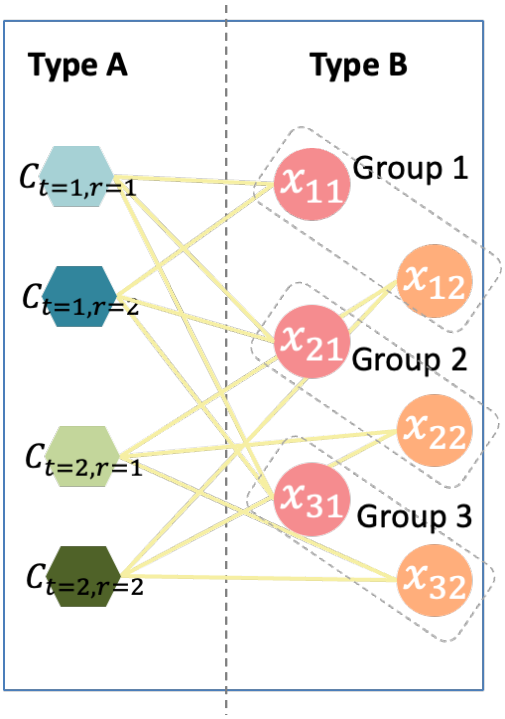

Figure 13. Decompose a symmetric graph for a capital budgeting instance: After applying the WL test, the graph contains two types of nodes: **Type-A**: uniquely colored nodes ($C_{t=1,r=1}, C_{t=1,r=2}, C_{t=2,r=1}, C_{t=2,r=2}$); **Type-B**: 3 groups of nodes that share the same color pattern. Each group contains nodes with distinct colors, the two groups have identical color distributions, and there are no edges between the groups.

As illustrated by the corresponding bipartite graph in Figure 13, the capital budgeting instance is symmetrically decomposable.

Example D.10 (Solver can be Inconsistant).

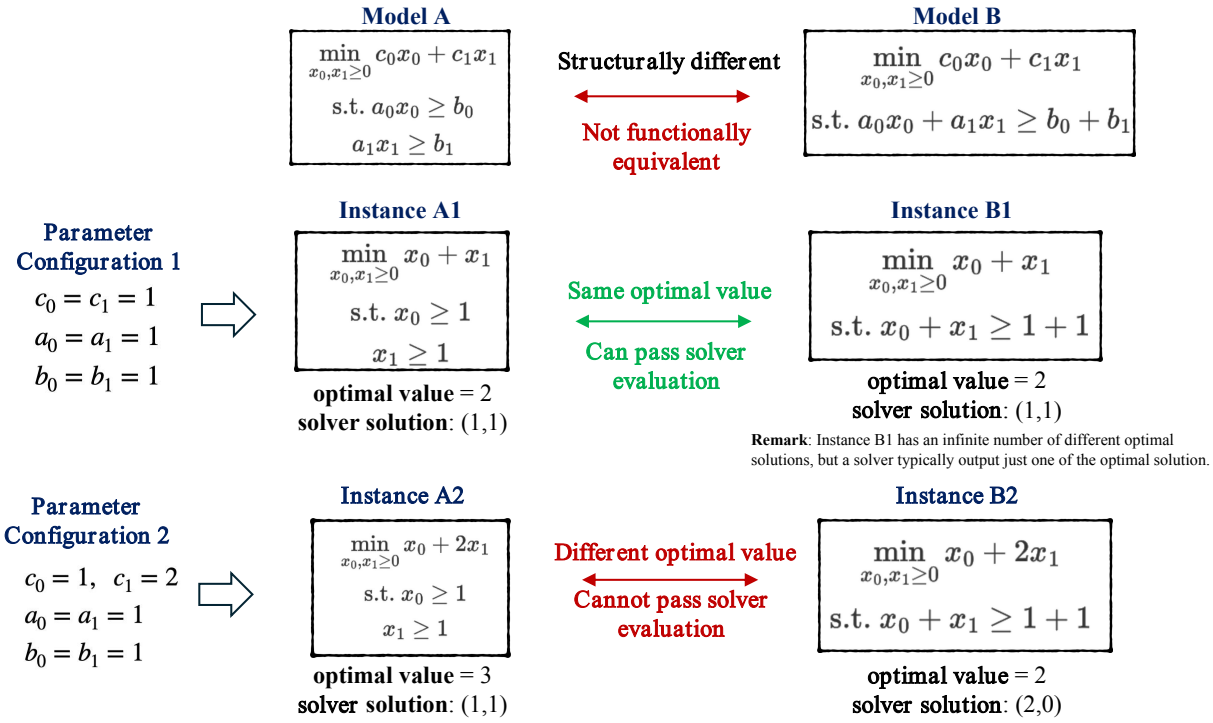

Figure 14. Solver's evaluation on modeling equivalence can be inconsistent across different parameter configurations

E. Error Analysis

Understanding why models fail is important for interpreting benchmark results beyond overall accuracy and compile-error rates. Rather than attempting to explain every model family in Table 3, we conduct a focused error analysis on two representative models, **GPT-4o** and **o1**. These two models exhibit a particularly informative contrast in the main benchmark: reasoning model like o1 produces fewer compile errors, whereas base model like GPT-4o achieves higher overall pass@1 accuracy comparing to the related reasoning model. This makes them suitable for analyzing why better code executability does not necessarily imply better optimization-modeling performance.

We annotate failed outputs using a two-level taxonomy: *compile errors* and *modeling errors*. Compile errors are failures that prevent successful execution and are divided into three categories:

- **Python syntax/semantic errors**: indentation mistakes, syntax errors, undefined names, and related Python-level failures.
- **Gurobi API misuse**: incorrect function names, invalid API calls, or malformed Gurobi-specific usage.
- **Problem/data-interface misunderstanding**: code errors caused by misinterpreting the role of inputs, parameters, indexing structure, or optimization components.

Modeling errors are mistakes in the optimization formulation itself:

- **Wrong objective**: the objective function does not match the intended optimization goal.
- **Wrong constraints**: one or more constraints are missing, incorrect, or logically mis-specified.
- **Wrong decision variables**: the generated variables have incorrect meanings, domains, indexing, or structural roles.

For each model, Table 7 reports the incidence of each error type among failed outputs. Note that modeling-error labels are **not mutually exclusive**: a single output may contain multiple formulation mistakes. Therefore, the percentages in Table 7 do not sum to 100.

Two patterns are especially clear. First, **o1 exhibits substantially fewer low-level compile errors than GPT-4o**. Python-related failures drop from 9.97% to 0.00%, Gurobi API errors from 5.30% to 0.40%, and problem/data-interface misunderstandings from 4.36% to 1.61%. However, this reduction in execution failures does *not* translate into higher end-to-end benchmark accuracy. Second, for *both* models, the dominant failure mode is **incorrect constraints**, accounting for 56.07% of failed outputs for GPT-4o and 60.08% for o1. Errors in decision-variable design are also common, and are more frequent for o1 (39.92%) than for GPT-4o (27.41%). By contrast, objective-function errors are relatively rare for both models.

Overall, this case study suggests that the main difficulty in Bench4Opt lies less in producing executable code and more in faithfully translating natural-language requirements into correct constraints and variable definitions. In other words, reducing compile errors alone is insufficient for strong optimization-modeling performance.

Table 7. Error analysis on Bench4Opt. Each percentage reports the fraction of *failed outputs* from a given model that exhibit the corresponding error type. Modeling-error categories are not mutually exclusive, so the percentages do not sum to 100.

Error Type	**GPT-4o**	**o1**
Python syntax / semantics	9.97	0.00
Gurobi API misuse	5.30	0.40
Problem/data-interface misunderstanding	4.36	1.61
Wrong objective	3.43	3.23
Wrong constraints	56.07	60.08
Wrong decision variables	27.41	39.92