Towards Human-aligned Evaluation for Linear Programming Word Problems

Linzi Xing[1*], Xinglu Wang[2*], Yuxi Feng[3], Zhenan Fan[1], Jing Xiong[4], Zhijiang Guo[5], Xiaojin Fu[5], Rindra Ramamonjison[1], Mahdi Mostajabdaveh[1], Xiongwei Han[5], Zirui Zhou[1], Yong Zhang[1]

[1]Huawei Technologies Canada, [2]Simon Fraser University, [3]University of British Columbia, [4]Sun Yat-sen University, [5]Huawei Noah's Ark Lab,
{linzi.xing, zhenan.fan1, guozhijiang, fuxiaojin, rindranirina.ramamonjison, mahdi.mostajabdaveh1, hanxiongwei, zirui.zhou, yong.zhang3}@huawei.com
xwa239@sfu.ca, fyx14@cs.ubc.ca, xiongj69@mail2.sysu.edu.cn

Abstract

Math Word Problem (MWP) is a crucial NLP task aimed at providing solutions for given mathematical descriptions. A notable sub-category of MWP is the Linear Programming Word Problem (LPWP), which holds significant relevance in real-world decision-making and operations research. While the recent rise of generative large language models (LLMs) has brought more advanced solutions to LPWPs, existing evaluation methodologies for this task still diverge from human judgment and face challenges in recognizing mathematically equivalent answers. In this paper, we introduce a novel evaluation metric rooted in graph edit distance, featuring benefits such as permutation invariance and more accurate program equivalence identification. Human evaluations empirically validate the superior efficacy of our proposed metric when particularly assessing LLM-based solutions for LPWP.

Keywords: Evaluation Methodologies, Language Modeling, NLP Application, Mathmatical NLP

1. Introduction

Math Word Problem (MWP), a fundamental yet challenging NLP task, has received considerable attention recently, aiming to provide solution expressions for given mathematical problem descriptions (Shen et al., 2021). Most prior research has focused on elementary arithmetic problems (Roy and Roth, 2015; Koncel-Kedziorski et al., 2016; Patel et al., 2021; Cobbe et al., 2021) and algebra problems (Kushman et al., 2014; Huang et al., 2016) due to their straightforward solvability and evaluative convenience through methods like precision/recall (Huang et al., 2016; Wang et al., 2017) and test solve rate (Cobbe et al., 2021). On the other hand, **Linear Programming Word Problems (LPWP)** (Ramamonjison et al., 2022), closely tied to real-world decision-making, hold significant potential in operations research (OR) (Tao et al., 2020; Beairsto et al., 2021; Fan et al., 2024) but remain under-explored. As shown in Figure 1, a LPWP typically consists of a textual problem description paired with its mathematical program, which comprises three main elements: *decision variables, an objective, and constraints*.

Previous solutions for LPWP often decompose this task into sub-steps (e.g., entity recognition followed by text generation), which leads to inevitable error accumulation (Ramamonjison et al., 2022; He et al., 2022; Prasath and Karande, 2023). Recently, generative large language models (LLMs) have emerged as more advanced, end-to-end alternatives for LPWP, leveraging problem descriptions as instructions to generate textual LPWP answers. Despite the rapid progress in LPWP task, its evaluation methodologies have not evolved at the same pace. Previous evaluation metrics (e.g., Canonical Accuracy (Ramamonjison et al., 2022) and Execution Accuracy (Prasath and Karande, 2023)) either do not align with human intuition or fail to recognize mathematically equivalent answers (§2). To address prior issues in LPWP evaluation, we introduce a simple yet effective evaluation strategy based on graph edit distance. Specifically, given the predicted (e.g., by LLMs) and reference (i.e., ground-truth) programs in the format of a textual answer, we initially parse them into a structured general form. We then convert these general forms into bipartite graphs, with weighted edges linking up vertices representing variables and constraints. Finally, we compute the edit distance between graphs

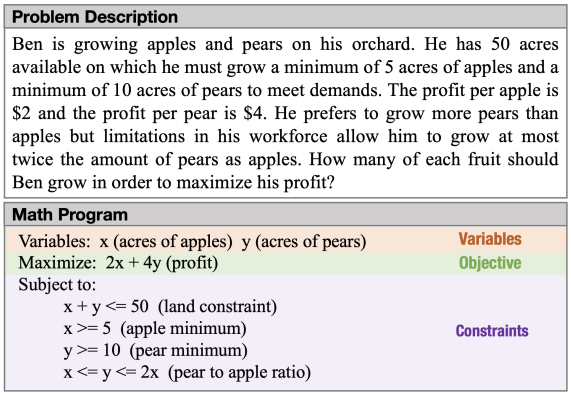

Figure 1: An LPWP example sampled from *NL4OPT* testing set (Ramamonjison et al., 2023b).

* The first two authors contributed equally to this work.

LREC-COLING 2024, pages 16550–16556
20-25 May, 2024.

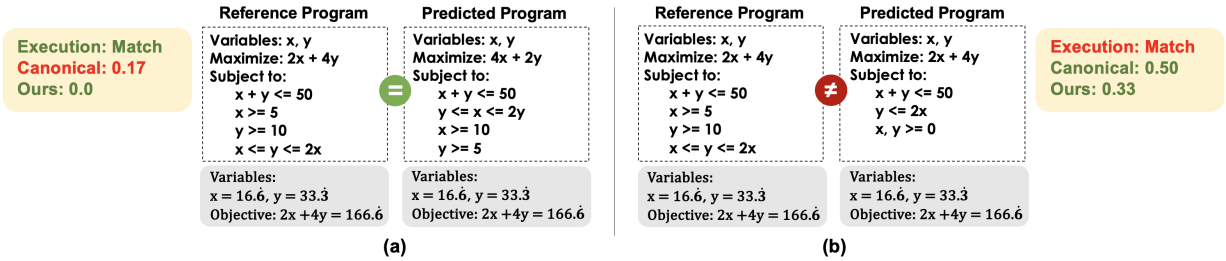

Figure 2: Demonstration of the pitfalls of *Canonical Accuracy* and *Execution Accuracy*: results highlighted in red are poorly aligned with human judgment.

corresponding to the predicted and reference programs, using this as the evaluation metric to assess the performance of modeling solutions for LPWP.

In summary, our proposed strategy showcases advancement in the following two aspects:

- **Permutation Invariance:** Our proposed graph-based evaluation strategy can accommodate order discrepancies of variables and constraints between the predicted and reference programs, as these order variations do not inherently indicate two programs are different.
- **Exact Match Identification:** Our proposal ensures that a match between predicted and reference programs can be confidently considered as exact equivalence. This addresses the challenge of the execution measurement (in §2), where two programs with the same (or an infeasible) optimal solution might still be inequivalent.

2. Pitfalls of Prior Metrics

Prior research typically employs two evaluation approaches to evaluate LPWP performance, namely *Canonical Accuracy* and *Execution Accuracy*:

Canonical Accuracy (Ramamonjison et al., 2022). This evaluation method is based on the declaration-level matching between predicted and reference programs, with a declaration representing either an optimization objective or constraint. In particular, the canonical accuracy for one LPWP problem can be calculated as follows:

$$Acc = 1 - \frac{\min(FP_i + FN_i, D_i)}{D_i} \quad (1)$$

where for a given problem i, D_i is the number of actual declarations in the reference program. The term false positives FP_i denotes the number of declarations in prediction not matched with any of the actual declarations, while false negatives FN_i denotes the number of actual declarations not matched with any of the predicted declarations. As $FP_i + FN_i$ can possibly exceed D_i, the $\min$ is leveraged to prevent negative Acc.

This canonical measurement strongly assumes that predicted programs must follow the same variable order as those in the ground-truth program. In other words, even though $a \cdot X + b \cdot Y \leq c$ and $a \cdot Y + b \cdot X \leq c$ (where X and Y permutes) are fundamentally equivalent, the canonical measurement still deem them as different due to the altered order of variables. As shown in Figure 2(a), the predicted program merely swaps variables and constraints compared to the reference one, which should be regarded as equivalent. However, the canonical metric yields a problematically low score, which ideally should be 1.0.

Execution Accuracy (Prasath and Karande, 2023). Similar to the standard evaluation for code generation that emphasizes functional correctness (Chen et al., 2021), the execution evaluation for LPWP measures the correctness of the mathematical program via comparing the optimal solutions between predicted and reference programs. Specifically, we parse the mathematical program in Fig 1 into an MPS[1] file format and then use a solver to obtain the optimal objective. Equal optimal objective values between the predicted and reference program indicate a successful LPWP prediction.

Nonetheless, this evaluation scheme cannot serve as a flawless indicator of program equivalence. As shown in Figure 2(b), even if the predicted program overlooks a considerable number of constraints, it may still match the reference's optimal value. Furthermore, two different programs both identified as "infeasible" by the solver will be mistakenly regarded as a match even the predicted program greatly differs from the reference one.

3. Evaluation via Graph Edit Distance

This section introduces a simple yet effective evaluation strategy grounded on the graph edit distance. This strategy tackles the pitfalls of prior metrics and aligns more closely with human sense, as a smaller edit distance indicates fewer mistakes in the predicted program *w.r.t* the reference program.

Generally, this evaluation strategy can be unfolded as three steps: **(1)** Converting the initial tex-

[1]An MPS (Mathematical Programming System) file is an industry-standard format for linear and mixed integer programs (Wikipedia, 2021) .

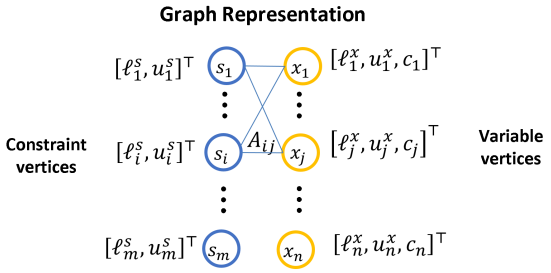

Figure 3: An example of graph representation.

tual form of the predicted and reference programs into Linear Programs (LPs) in *general forms*; **(2)** Transforming the predicted and reference LPs into *bipartite graphs*; **(3)** Calculating the *graph edit distance* between the predicted and reference graphs.

LP in General Form. The general form is widely adopted by various LP solvers including CPLEX (Cplex, 2009) and Gurobi (Gurobi Optimization, LLC, 2022). Formally, an LP with n variables and m constraints can be represented as:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \boldsymbol{\ell}^s \leq \mathbf{A}\mathbf{x} \leq \mathbf{u}^s \\ & \boldsymbol{\ell}^x \leq \mathbf{x} \leq \mathbf{u}^x, \end{aligned} \tag{P}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is the constraint matrix, $\mathbf{c} \in \mathbb{R}^n$ is the cost vector, $\mathbf{x} \in \mathbb{R}^n$ is the decision variables. Extended real domains are denoted by $\underline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$ and $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$. $\boldsymbol{\ell}^x \in \underline{\mathbb{R}}^n$ and $\mathbf{u}^x \in \overline{\mathbb{R}}^n$ are lower/upper bound of the decision variable $\mathbf{x}$, and $\boldsymbol{\ell}^s \in \underline{\mathbb{R}}^m$ and $\mathbf{u}^s \in \overline{\mathbb{R}}^m$ are lower/upper bound of the constraint. The type of constraint includes equality, two-sides or one-side inequality. For the one-side inequality constraint, we prefer the right-side over the left-side one, by multiplying a constant -1.

For this step, we implement a robust rule-based parser to convert the initial textual math program into the LP in such general form.

Graph representation. It is well known that LP can be represented as an attributed bipartite graph, (Gasse et al., 2019; Fan et al., 2023), denoted as $\mathcal{G} = (S \cup X, E)$ (Figure 3). This graph consists of two disjoint vertex sets $S = \{s_i | i \in [m]\}$ and $X = \{x_j | j \in [n]\}$, and a collection $E = \{e_{ij} | i \in [m], j \in [n]\}$ of edges. Here, notation $[\cdot]$ means a set of consecutive numbers. Vertex s_i corresponds to the i-th constraint $\ell_i^s \leq \mathbf{a}_i^\top \mathbf{x} \leq u_i^s$, with its attribute being $\text{attr}(s_i) = [\ell_i^s, u_i^s]^\top$. The notation x_j is overloaded in the graph context to represent the vertex x_j that corresponds to the decision variable x_j. Its attribute $\text{attr}(x_j) = [\ell_j^x, u_j^x, c_j]^\top$ contains the bounds (ℓ_j^x, u_j^x) and objective coefficient (c_j). The topology of $\mathcal{G}$ is determined by $\mathbf{A}$, *i.e.*, edge e_{ij} exists iff $A_{ij} \neq 0$. The attribute of this edge is simply the weight A_{ij}, *i.e.*, $\text{attr}(e_{ij}) = [A_{ij}]$.

One significant advantage of representing LP as an attributed bipartite graph is its ***permutation invariance***. It refers that two LPs are equivalent even if the constraints are permuted, or if the decision variables (and correspondingly, the cost vector and columns in the matrix $\mathbf{A}$) are permuted. Using this bipartite graph representation, we can uniquely convert any LP in general form into an attributed bipartite graph.

Graph edit distance (GED). GED is the minimum cost required to transform one graph into another by a sequence of operations including inserting, deleting, and substituting vertices and/or edges (as shown in Fig 4). For generality, all these operations are viewed as matching, *e.g.*, deleting vertex is to match this vertex to an empty vertex, denoted by ϵ.

Any well-established GED algorithm (Abu-Aisheh et al., 2015; Riesen et al., 2020; Gao et al., 2010) can be employed once the costs of matching operations are determined. Although various cost definitions can exist, our proposed definition abides by a straightforward principle: each operation on an attribute in graph incurs a unit cost of 1. Following this principle and given the graph of predicted program $\mathcal{G}^p = (S^p \cup X^p, E^p)$ and the graph of reference program $\mathcal{G}^r = (S^r \cup X^r, E^r)$, the vertex cost matrix $\mathbf{C}_v$ is formally defined as:

	$s_{i'}^r \in S^r$	$x_{i'}^r \in X^r$	ϵ
$s_i^p \in S^p$	#msm$(s_i^p, s_{i'}^r)$	∞	#attr(s_i^p)
$x_i^p \in X^p$	∞	#msm$(x_i^p, x_{i'}^r)$	#attr(s_i^p)
ϵ	#attr$(s_{i'}^r)$	#attr$(s_{i'}^r)$	∞

Due to the space limit, we details only a few entries highlighted in gray above. (1) Substituting constraint vertex s_i^p to $s_{i'}^r$ equals to editing attributes of one vertex to match another, so the cost of such operation is the number of mismatched attributes between two vertices, denoted by $\mathbf{C}_v(s_i^p \to s_{i'}^r) =$ #msm$(s_i^p, s_{i'}^r)$. Given that GED accounts for the interaction between vertices and edges, here only current pairs of vertices need consideration. (2) Regardless of how attributes are edited, a constraint vertex will never convert to a variable vertex, so their substitution cost is ∞. (3) The deletion cost of a constraint vertex s_i^r equals to the number of its attributes, *i.e.*, $\mathbf{C}_v(s_i^r \to \epsilon) =$ #attr(s_i^r). Analogously, the insertion cost can be deemed as converting an empty vertex ϵ into the vertex being inserted.

Similarly, the edge cost matrix $\mathbf{C}_e$ is defined as:

	$e_{ij}^r \in E^r$	ϵ
$e_{ij}^p \in E^p$	#msm(e_{ij}^p, e_{ij}^r)	#attr(e_{ij}^p)
ϵ	#attr(e_{ij}^r)	∞

While GED$(\mathcal{G}^p, \mathcal{G}^r)$ can measure the similarity between predicted and reference programs, it is sensitive to graph size. A larger graph, representing a predicted program with more variables and constraints, is more prone to errors, thereby leading to larger GED *w.r.t* the reference program. To address this issue, we further normalize GED$(\mathcal{G}^p, \mathcal{G}^r)$ by the graph size, as NGED$(\mathcal{G}^p, \mathcal{G}^r) = \frac{\text{GED}(\mathcal{G}^p, \mathcal{G}^r)}{\max(|\mathcal{G}^p|, |\mathcal{G}^r|)}$, where $|\mathcal{G}| = \sum_{e\in E} \#\text{attr}(e) + \sum_{v\in S\cup X} \#\text{attr}(v)$. Ultimately, NGED forms the core of our proposed evaluation metric for LPWP. Furthermore, graphs

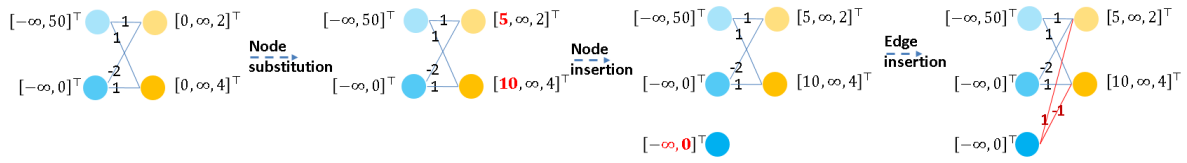

Figure 4: Exemplar graph edit path from the graph associated with the predicted program to the reference program in Figure 2(b). Blue and yellow vertices are respectively constraint and variable vertices.

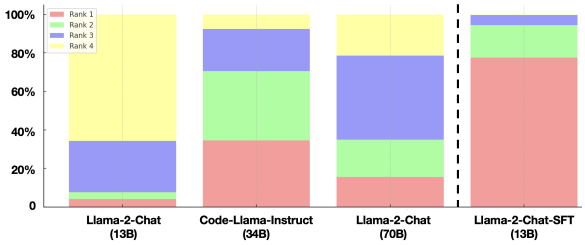

Figure 5: Ranking distributions of human judgements on the NL4OPT test set.

Language Models	Execution(↑)	Canonical(↑)	Ours(↓)
Llama-2-Chat (13B)	0.07 (4)	0.24 (4)	0.52 (4)
Code-Llama-Instruct (34B)	0.35 (2)	0.54 (2)	0.25 (2)
Llama-2-Chat (70B)	0.21 (3)	0.31 (3)	0.41 (3)
Llama-2-Chat-SFT (13B)	0.53 (1)	0.64 (1)	0.14 (1)

Table 1: Evaluation scores on test set via 3 metrics for 4 models. ↑ means larger is better. ↓ means lower better. Models' ranks are in brackets.

of larger scale also require the scalability of our GED-based evaluation metric to be both stable and robust. Therefore, we discuss the computational complexity of our metric in detail in Appendix A.

4. Experiments and Analysis

4.1. Experimental Setup

Datasets. We employ the recently introduced NL4OPT (Ramamonjison et al., 2022), the first-ever LPWP dataset, in our experiments. This dataset contains 713 training, 99 validation, and 289 testing data points. Each data point consists of both a problem description and a human-composed mathematical program (as the example in Figure 1).

Language Models. To assess the effectiveness of evaluation metrics comprehensively, we consider four LLMs, all rooted in the foundational architecture of the widely-explored, open-sourced Llama family (Touvron et al., 2023; Rozière et al., 2023) but with different settings to obtain diverse LPWP modeling. Specifically, we include three Llama-based models: (1) **Llama-2-Chat (13B)**, (2) **Code-Llama-Instruct (34B)**, and (3) **Llama-2-Chat (70B)**. Additionally, we also fine-tune Llama-2-Chat (13B) with the training set of NL4OPT and name it as **Llama-2-SFT (13B)**. Except for Llama-2-SFT (13B), all three other LLMs are under the one-shot in-context learning (ICL) setting, where a validation datapoint is randomly selected and utilized as the one-shot example for all inferences. More details about the prompt template we used can be found in Appendix B.

Human Evaluation. We collected human judgments from three OR experts within our institution. The annotation task is structured as follows: for each test sample, an annotator is provided with its reference program and 4 anonymized programs predicted by the aforementioned 4 LLMs. The annotator then compares predicted programs with the reference program and ranks them in descending order according to their deviation from the reference program (e.g., $LLM_1 = LLM_2 > LLM_4 > LLM_3$). It is worth noting that annotators have the option to use "=" if two predicted programs appear equally similar to the reference program. As shown in Figure 5, four LLMs exhibit diverse performance and the performance ranking by human is "*llama-13b-sft* > *code-llama-34b* > *llama-70b* > *llama-13b*".

4.2. Experimental Results

Performance of LLMs. Table 1 presents the evaluation results obtained from our proposed graph-based metric alongside two baseline metrics (i.e., execution accuracy and canonical accuracy in §2), averaged across 289 test samples . The rankings of LLMs based on these three metrics are consistent with human judgment shown in Figure 5, indicating that all three evaluation metrics can effectively assess language models' capability to solve LPWP to some extent. However, this does not suggest that they align equally well with human judgement. For the majority of test samples, which are either distinctly easy or challenging for specific LLMs, the discrepancies between their predicted and reference programs can be easily quantified by all metrics.

Correlation with Human Evaluation. To more comprehensively measure the alignment between human evaluation and three automatic evaluation metrics for LPWP, we delve deeper by looking into the ranking match for each test sample. Specifically, we define two types of matching rates: coarse-grained rate (**C-Match**) and fine-grained rate (**F-Match**). Given two ranking lists obtained by human judgment and the automatic metric, we call it "lists exactly match" if these two ranking lists are iden-

Metrics	C-Match	F-Match
Execution	9 / 289	716 / 1734
Canonical	64 / 289	1336 / 1734
Ours	**178 / 289**	**1641 / 1734**

Table 2: Ranking match rate between automatic evaluation metrics and human judgements.

tical. the C-Match measures the percentage of instances where the human and automatic ranking lists exactly match. On the other hand, the F-match decomposes ranking lists into individual ranking pairs and then calculates the match rate at the pair level. As shown in Table 2, our proposed evaluation metric consistently achieves the highest match rate with human evaluations at both granularities. This highlights the enhanced reliability and alignment with human judgment of our proposal, especially when conducting evaluation in a pairwise manner (comparing merely two models LLM_1 and LLM_2).

5. Conclusion and Future Work

In this paper, we present a graph-based evaluation metric for LPWP, emphasizing permutation invariance and exact match identification. Experiments show superior alignment with human evaluation. In future work, we aim to extend this metric to other mathematical programming word problems like quadratic and mixed-integer ones, which can also be graph-represented. Furthermore, we also tend to use this metric as a reward function to enhance LLMs' RLHF training for such problems and to incorporate it into relevant operations research modeling applications (Ramamonjison et al., 2023a), thereby enhancing their efficacy and utility. Additionally, we intend to utilize our proposed metric to conduct a more thorough evaluation of LLMs' performance on LPWPs, particularly in in-context settings (Xiong et al., 2024).

6. Bibliographical References

Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. 2015. An exact graph edit distance algorithm for solving pattern recognition problems. In *ICPRAM*, pages 271–278. SciTePress.

Jeneva Beairsto, Yufan Tian, Linyu Zheng, Qunshan Zhao, and Jinhyun Hong. 2021. Identifying locations for new bike-sharing stations in glasgow: an analysis of spatial equity and demand factors. *Annals of GIS*, 0(0):1–16.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

IBM ILOG Cplex. 2009. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157.

Zhenan Fan, Bissan Ghaddar, Xinglu Wang, Linzi Xing, Yong Zhang, and Zirui Zhou. 2024. Artificial intelligence for operations research: Revolutionizing the operations research process. *CoRR*, abs/2401.03244.

Zhenan Fan, Xinglu Wang, Oleksandr Yakovenko, Abdullah Ali Sivas, Owen Ren, Yong Zhang, and Zirui Zhou. 2023. Smart initial basis selection for linear programs. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 9650–9664. PMLR.

X. Gao, B. Xiao, D. Tao, et al. 2010. A survey of graph edit distance. *Pattern Anal Applic*, 13:113–129.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. 2019. Exact combinatorial optimization with graph convolutional neural networks. *NeurIPS*, 32.

Gurobi Optimization, LLC. 2022. Gurobi Optimizer Reference Manual.

JiangLong He, Mamatha N, Shiv Vignesh, Deepak Kumar, and Akshay Uppal. 2022. Linear programming word problems formulation using ensemblecrf ner labeler and t5 text generator with data augmentations.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 887–896, Berlin, Germany. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland. Association for Computational Linguistics.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Ganesh Prasath and Shirish Karande. 2023. Synthesis of mathematical programs from natural language specifications.

Rindra Ramamonjison, Haley Li, Timothy Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-dehkordi, Zirui Zhou, and Yong Zhang. 2022. Augmenting operations research with auto-formulation of optimization models from problem descriptions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 29–62, Abu Dhabi, UAE. Association for Computational Linguistics.

Rindra Ramamonjison, Timothy Yu, Linzi Xing, Mahdi Mostajabdaveh, Xiaorui Li, Xiaojin Fu, Xiongwei Han, Yuanzhe Chen, Ren Li, Kun Mao, and Yong Zhang. 2023a. LaTeX2Solver: a hierarchical semantic parsing of LaTeX document into code for an assistive optimization modeling application. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 471–478, Toronto, Canada. Association for Computational Linguistics.

Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. 2023b. Nl4opt competition: Formulating optimization problems based on their natural language descriptions.

Kaspar Riesen and Horst Bunke. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959. 7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007).

Kaspar Riesen, Miquel A. Ferrer, and H. Bunke. 2020. Approximate graph edit distance in quadratic time. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code.

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. 2021. Generate & rank: A multi-task framework for math word problems. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2269–2279, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Diana Qing Tao, Martin Pleau, et al. 2020. Analytics and Optimization Reduce Sewage Overflows to Protect Community Waterways in Kentucky. *Interfaces*, 50(1):7–20.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor

Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

Wikipedia. 2021. MPS (format) – Wikipedia, The Free Encyclopedia.

Jing Xiong, Zixuan Li, Chuanyang Zheng, Zhijiang Guo, Yichun Yin, Enze Xie, Zhicheng Yang, Qingxing Cao, Haiming Wang, Xiongwei Han, Jing Tang, Chengming Li, and Xiaodan Liang. 2024. Dq-lore: Dual queries with low rank approximation re-ranking for in-context learning.

A. Discussion about Computational Complexity

We adopted the exact algorithm proposed in Abu-Aisheh et al. (2015), which is a branch-and-bound algorithm with tailored pruning and branching strategy. Given two bipartite graphs with $n1$ and $n2$ nodes respectively, the worst-case complexity is $O((n1 * n2)^{n1+n2})$. Practically, as the predicted and ground-truth graphs usually share similar characteristics, such as easily identifiablely identical nodes, the algorithm scales well, e.g., processing graphs with 15 nodes within 60 seconds. Admittedly, a limitation is its inability to scale to larger numbers of nodes. However, as mentioned in section 3, we can benefit from various existing algorithms since we frame the computation of the evaluation metric as a graph edit distance problem. One possible solution is to use an approximate algorithm, such as the one in Riesen and Bunke (2009) with cubic complexity $O(max\{n1, n2\}^3)$. However, this topic and a more detailed exploration are left for future work.

B. Prompt Templates

Figure 6 is the complete prompting formulation that we use identically across all four LLMs in our experiments (section 4.1).

Content of Prompt

[TEXT OF PROBLEM DESCRIPTION]

Use the above problem descriptions and write the optimization formulation of the problem. Please only give me the model with just one line of explanation for each model element. I don't need the solution. Remove all non-essential spaces. Don't simplify the expressions and don't use latex code or any code in your responses. Use 'x', 'y' and 'z' as variable names.

Please give me the model with the format following the format of the example given below, please do not include any content other than model in your answer:

[RANDOMLY SAMPLED ONE-SHOT EXAMPLE]

Figure 6: The prompt templates we applied for four Llama-based language models in section 4.1. The randomly sampled one-shot example is not added for Llama-2-SFT (13B).