BPP-Search: Enhancing Tree of Thought Reasoning for Mathematical Modeling Problem Solving

Anonymous ACL submission

Abstract

LLMs exhibit advanced reasoning capabilities, offering the potential to transform natural language questions into mathematical models. However, existing open-source datasets in operations research domain lack detailed annotations of the modeling process, such as variable definitions, focusing solely on objective values, which hinders reinforcement learning applications. To address this, we release the **StructuredOR** dataset, annotated with comprehensive labels that capture the complete mathematical modeling process. We further propose **BPP-Search**, a algorithm that integrates reinforcement learning into a tree-of-thought structure using **B**eam search, a **P**rocess reward model, and a pairwise **P**reference algorithm. This approach enables efficient exploration of tree structures, avoiding exhaustive search while improving accuracy. Extensive experiments on StructuredOR, NL4OPT, and MAMO-ComplexLP datasets show that BPP-Search significantly outperforms state-of-the-art methods. In tree-based reasoning, BPP-Search excels in accuracy and efficiency, enabling faster retrieval of correct solutions.

1 INTRODUCTION

Mathematical modeling, particularly Linear Programming (LP) and Mixed Integer Programming (MIP), plays a critical role in industrial applications such as logistics (Demirel and Gökçen, 2008), electricity scheduling and transmission (Zhang et al., 2018), and supply chain management (Özceylan and Paksoy, 2013). With the advent of Large Language Models (LLMs), transforming natural language questions into mathematical models has become a promising approach for automating operations research tasks (Xiao et al., 2024; Tang et al., 2024a).

Despite the increasing availability of open-source operations research (OR) datasets designed for question-to-model transformation (Huang et al., 2024; Ramamonjison et al., 2022; Tang et al., 2024b), these datasets primarily focus on objective values while lacking detailed annotations of the underlying modeling processes. This gap limits the application of Reinforcement Learning (RL), as prior studies (Lightman et al., 2023; Uesato et al., 2022; Cobbe et al., 2021) have shown that process information can significantly enhance mathematical reasoning performance. To address this limitation, we design a rigorous framework for dataset generation and introduce the **StructuredOR** dataset, which not only provides objective values for evaluation but also includes comprehensive annotations of the modeling process, enabling broader applicability in RL-based methods.

Chain-of-Thought (CoT) (Wei et al., 2022), Self-Consistency (SC) (Wang et al., 2023) and Tree-of-Thought (ToT) (Yao et al., 2023) have demonstrated substantial improvements in reasoning tasks. However, these approaches have inherent limitations. CoT heavily relies on the policy model and generates only one reasoning path at a time, making it likely to fail to find the correct answer when the policy model is weak. SC, without a verifier, struggles to validate the correctness of candidate answers, allowing errors in intermediate steps to propagate and mislead the reasoning process. Similarly, ToT generates multiple leaf nodes as potential answers, but without a verifier, it is unclear which leaf node should be selected as the final solution. Nonetheless, ToT remains promising; with sufficiently wide and deep trees and effective node selection strategies, it has the potential to generate optimal solutions.

To enhance the reasoning process within the ToT framework, we propose **BPP-Search**, a novel method that integrates **B**eam Search, a **P**rocess Reward Model (PRM), and a pairwise **Preference** algorithm. BPP-Search is designed to improve accuracy and reduce unnecessary node exploration, making it particularly effective for complex reason-

ing tasks in mathematical modeling.

Our contributions are threefold: (1) We introduce the **StructuredOR** dataset, which bridges the gap between existing datasets and the requirements of RL-based methods by providing detailed modeling annotations. (2) We propose **BPP-Search** and explore heuristic algorithms combined with PRM, including Beam Search (Lowerre and Reddy, 1976), Greedy (Prim, 1957), Epsilon Greedy (Sutton, 2018), and Random Greedy we proposed in Section 4.3. (3) We conduct extensive experiments on the **StructuredOR**, **NL4OPT** (Ramamonjison et al., 2022), and **Mamo-ComplexLP** (Huang et al., 2024) datasets, demonstrating the superiority of BPP-Search over baseline and current state-of-the-art methods from the perspective of efficiency and accuracy.

2 RELATED WORK

2.1 Mathematical Modeling Datasets

Mathematical modeling datasets can be broadly categorized into two types: abstract modeling and concrete instance modeling. Modeling tools such as Pyomo (Hart et al., 2017) and AMPL (Gay, 2015), and OPL (Van Hentenryck et al., 1999) provide support for both approaches, enabling users to work with abstract models as well as concrete instances.

Abstract modeling focuses on capturing the essential structural information of a model. It typically involves two steps: defining basic model declarations and applying data to create concrete instances. This approach is particularly suited for large-scale industrial applications and research, as models can be defined once and reused by importing different datasets. For instance, ML-Prompt (Wang et al., 2024b) leverages abstract models to generate parameter distributions, which are subsequently populated with specific values to construct concrete instances within industrial pipelines. Additionally, several studies (Yang et al., 2024c; Wang et al., 2024c) combine abstract models with CoT and LLMs to address problems such as Traveling Salesman Problem (Gavish and Graves, 1978), bypassing the need for traditional mathematical solvers or explicit concrete models.

Concrete modeling requires all data to be available before model processing begins, making it a straightforward and efficient approach. It is particularly suited for analytical projects. This approach is especially advantageous when certain constraints are difficult or time-consuming to generalize into an abstract format, as it allows for more precise and tailored solutions, significantly reducing processing time. For smaller-scale mathematical models, tasks can be solved directly without treating them as a combination of abstract modeling and Named Entity Recognition (Grishman and Sundheim, 1996), which involves first building an abstract model and then mapping numerical parameter values to it. This approach minimizes error accumulation across tasks (Shen et al., 2023).

2.2 Process Reward Model

Several works (Cobbe et al., 2021; Lightman et al., 2023; Uesato et al., 2022) introduced the concept of the Process Reward Model (PRM), demonstrating its ability to significantly enhance the performance of weak and small-scale policy models. Compared to the Outcome Reward Model (ORM), PRM achieves better performance but incurs higher labeling costs (Uesato et al., 2022). Initially, PRM training relied on manually labeled data (Lightman et al., 2023). To address the growing demand for processing labels, Monte Carlo Tree Search (MCTS)-based methods (Wang et al., 2024d,a; Luo et al., 2024; Zhang et al., 2024; Setlur et al., 2024) were later developed to simulate and assign scores to reasoning processes. While effective, MCTS-based approaches require wide and deep trees to generate labeled data through extensive rollouts for score convergence at intermediate nodes, resulting in extremely high computational resource demands. In contrast, manually labeled data is deterministic and directly reflects the intended reasoning process without relying on approximations.

3 Dataset Generation

3.1 Preliminary

Greedy. The Greedy algorithm selects the candidate with the highest score at each step, focusing entirely on exploitation without exploration. The selection process can be formalized as:

$$a^* = \arg\max_{a \in A} P(a), \tag{1}$$

where $P(a)$ represents the score of candidate a, and A denotes the set of candidates.

Epsilon Greedy. The Epsilon Greedy algorithm balances exploration and exploitation during candidate selection. At each step, with a probability of ϵ, the algorithm selects a candidate randomly.

Otherwise, it selects the candidate with the highest score. The selection process can be formalized as:

$$a^* = \begin{cases} \text{random choice}, & w.p. \quad \epsilon, \\ \arg\max_{a \in A} P(a), & w.p. \quad 1-\epsilon, \end{cases} \tag{2}$$

where $P(a)$ represents the score of candidate a, and A denotes the set of candidates. The parameter ϵ controls the probability of exploring randomly versus exploiting the best-known option.

Beam Search. Beam Search is a heuristic search algorithm that explores a fixed number (k) of the most promising candidates (beam width) at each step. Unlike Greedy, which selects only the best candidate, Beam Search maintains a set of top k candidates to balance exploration and exploitation. The selection process can be formalized as:

$$B_{t+1} = \text{Top-}k \left(\bigcup_{a \in B_t} \text{Expand}(a) \right), \tag{3}$$

where B_t represents the set of beam candidates at step t, Expand(a) denotes the set of all possible successors of candidate a, and Top-k selects the k candidates with the highest scores. The parameter k controls the trade-off between computational cost and search completeness.

Tree of Thought. Compared with CoT and SC, ToT has the potential to generate many accurate results, but it faces challenges in selecting a single answer from numerous leaf nodes. Our objective is to accelerate this process and efficiently retrieve a satisfactory solution. In the mathematical modeling task, Fig. 1 illustrates the reasoning process, following a structured path through the question, sets, parameters, variables, objectives, and constraints (see Appendix A.3 for an example). Constructing a six-layer tree for each example across all datasets incurs high computational costs. To address this, we group nodes based on property similarities and limit each node to a maximum of three child nodes. The resulting tree structure is as follows: the first layer represents the question, the second combines sets and parameters, the third includes variables, and the fourth integrates objectives and constraints, as shown in Fig. 2. This approach balances tree width and computational efficiency for experiments.

3.2 StructuredOR Dataset Framework

Existing operations research datasets predominantly focus on objective values and the annotations of the underlying modeling process appear to be missing. To bridge the gap, we introduce **StructuredOR** a new dataset explicitly designed to provide the objective value for evaluation and capture the complete mathematical modeling process.

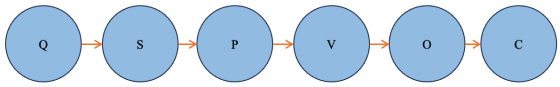

Figure 1: Reasoning steps. The process follows the path $Q \rightarrow S \rightarrow P \rightarrow V \rightarrow O \rightarrow C$, where Q, S, P, V, O and C represent the question, set, parameter, variable, objective, and constraint respectively.

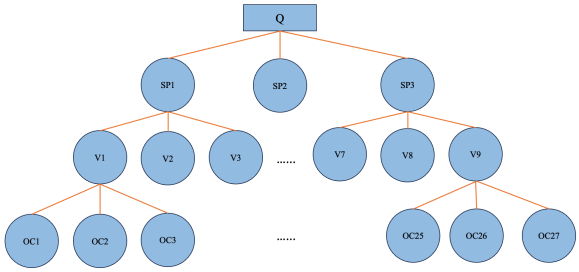

Figure 2: The structure of the Tree of Thought. Here, Q represents the question, SP represents set and parameter, V represents variable, and OC represents objective and constraint.

Building on Xiao's work (Xiao et al., 2024), which introduces a framework for generating abstract mathematical models, we refine and expand this approach to cover a wider spectrum of abstract models. We leverage LLM such as GPT-4o (Achiam et al., 2023) in conjunction with mathematical solvers to instantiate abstract models into concrete examples. Furthermore, we implement a series of validation mechanisms to construct and verify the accuracy of these concrete problems, thereby ensuring the dataset's quality and reliability. In summary, the StructuredOR dataset provides pairs of concrete questions and corresponding model data, accompanied by comprehensive annotations detailing the entire modeling process. Fig. 3 illustrates the construction pipeline, with each step distinguished by a color-coded arrow: blue for Step 1, orange for Step 2, red for Step 3, and green for Step 4. The whole process is delineated as follows:

Firstly, we leverage LLMs such as GPT-4o to generate distributions for sets and parameters in abstract models, inspired by prior work (Wang et al., 2024b) demonstrating that LLMs, such as GPT-4o, are capable of producing realistic distributions

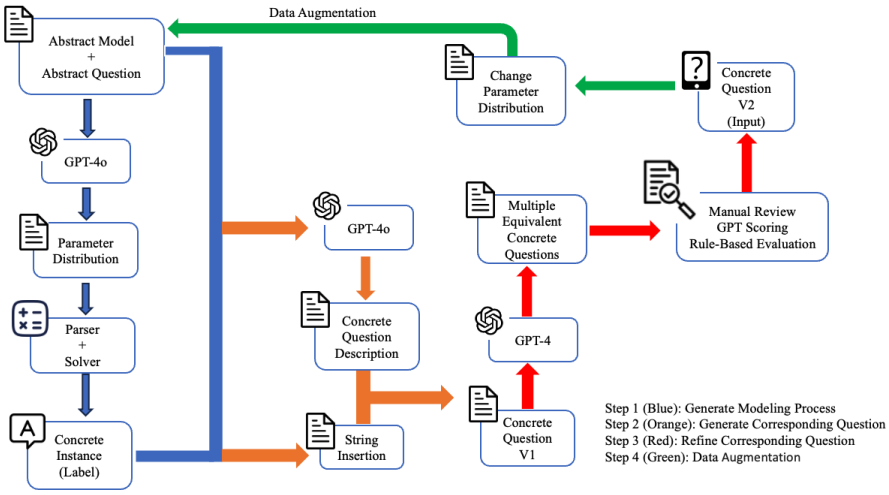

Figure 3: Pipeline of the construction process of our proposed StructuredOR dataset.

for such applications. This is followed by a simulation process to generate instance-specific parameter data. Subsequently, these parameters are converted into concrete models in LP format using a parser and a modeling tool. The resulting models are then validated for solvability and correctness using the Gurobi solver (Achterberg, 2019). Solvable problems from the solver are then selected as labeled instances, encompassing both the modeling process and the associated objective values. Details about standardizing the mathematical modeling data format are provided in Appendix A.1.

Next, based on the descriptions of sets and parameters in abstract models, we construct templates to generate markdown-formatted lists via string insertion to describe the information of sets and parameters. We subsequentially employ GPT-4o to develop a concrete problem description in natural language, which does not involve specific values for sets or parameters but provides a contextualized description of the problem. By concatenating this description with the numerical details of sets and parameters, a complete problem statement is produced. This string-based insertion ensures consistency and accuracy in the generated problem descriptions.

We further utilize GPT-4o to rephrase each generated question into three semantically equivalent versions to enhance fluency and naturalness. This process is supported by a rigorous review framework, comprising manual filtering, GPT-based scoring, and rule-based evaluation, to ensure semantic consistency between the rephrased texts and their corresponding labels.

Finally, we introduce an iterative strategy for data augmentation by changing parameter data distributions during the initial generation phase. This yields a dataset comprising 124 concrete questions and their corresponding models, spanning domains including logistics, scheduling, and networks, of which 77 examples are from the original framework, and 47 are generated through data augmentation.

Appendix A.2 provides an example of a concrete question along with its structured modeling process as the label. It also discusses the limitations of other datasets, such as Mamo-ComplexLP (Huang et al., 2024), NL4OPT (Ramamonjison et al., 2022), and IndustryOR (Tang et al., 2024b), highlighting their challenges in addressing the complete modeling process. Since most datasets, apart from StructuredOR, are incompatible with the schema defined in Appendix A.1, many questions in these datasets cannot be successfully parsed, rendering some examples unusable. Given the small size of the IndustryOR (Tang et al., 2024b) dataset, our experiments focus on the StructuredOR, Mamo-ComplexLP, and NL4OPT datasets.

3.3 PRM Dataset Preparation

To implement process supervision within the tree structure, we introduce the PRM (Uesato et al., 2022; Lightman et al., 2023), which assigns a score to each intermediate step in the reasoning process. There are two main approaches to generating training data for PRMs. (Uesato et al., 2022; Lightman et al., 2023) rely on manually labeling each intermediate step while (Wang et al., 2024d,a; Luo et al., 2024; Zhang et al., 2024; Setlur et al., 2024) leverage MCTS to assign scores to intermediate steps.

MCTS-based methods rely on wide and deep trees to generate labeled process data through extensive rollouts, ensuring score convergence for intermediate nodes but demanding significant computational resources. In contrast, manually labeled data is deterministic and directly captures the intended reasoning process without relying on approximations.

We first utilize the CoT and ToT frameworks across various policy models, including GPT (Achiam et al., 2023), LLama (Dubey et al., 2024), and Qwen series (Yang et al., 2024b), applied to the NL4Opt (Ramamonjison et al., 2022) and MAMO-ComplexLP (Huang et al., 2024) training datasets. Examples with consistent objective values are assumed to have correct modeling processes and therefore do not require manual labeling. Although the StructuredOR dataset already includes detailed modeling process annotations and serves as a source of high-quality positive examples, we adopt the same procedures to further augment its training data.

The process of modeling labels for both correct and incorrect reasoning paths is structured in alignment with the layers of the ToT framework. This follows a cumulative approach, where each layer is constructed sequentially, building upon the outcomes of the preceding layer. If the generated label represents a correct reasoning path, all segmented labels derived from it are also correct, as the correctness of each segment ensures the validity of the entire path. Conversely, if the overall reasoning path is determined to be incorrect and a specific intermediate step can be definitively identified as incorrect, all subsequent steps from that point onward are also labeled as incorrect, as errors propagate forward in the reasoning process. If no specific intermediate step can be identified as incorrect, the entire reasoning path is simply labeled as incorrect without making assumptions about the correctness or incorrectness of individual intermediate steps.

To diversify the PRM dataset, manual perturbations are applied to enrich it with diverse examples in both categories. Detailed descriptions of the PRM training dataset preparation are provided in Appendix A.4.

4 Methodology

4.1 Training PRM

Previous works (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2023; Luo et al., 2024) have shown that small-scale LLMs equipped with verifiers evaluating intermediate processes are capable of outperforming foundational large-scale LLMs in mathematical reasoning tasks. In this work, we fine-tune Qwen2.5-Math-1.5B (Yang et al., 2024b) for a binary classification task. Details on constructing prompts for the PRM are provided in Appendix A.5. After full-parameter supervised fine-tuning, we extract the logits corresponding to the correct label and apply the sigmoid function to compute the score:

$$S_{\text{PRM}} = \frac{1}{1 + e^{-l_{prm}}}, \tag{4}$$

where l_{prm} denotes the logit value for the correct label, and S_{PRM} represents the PRM score.

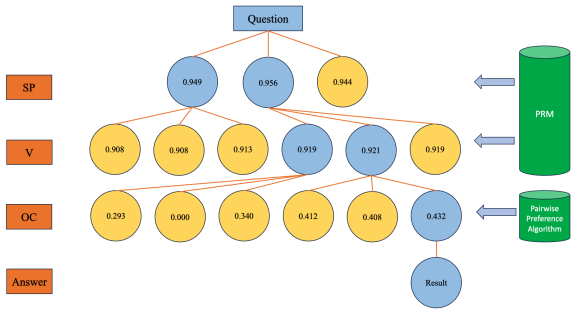

Figure 4: A real demonstration of the BPP-Search process with a beam search width of 2. Yellow nodes represent pruned nodes that are not explored, while light blue nodes indicate nodes that have been visited.

4.2 BPP-Search

We integrate the PRM with Greedy (Prim, 1957) and Beam Search (Lowerre and Reddy, 1976) algorithms, where the PRM provides scores to guide node selection. However, as shown in Section 5.3, increasing the Beam Search width does not consistently improve performance and, in some cases, leads to degradation. This limitation stems from the PRM, which is trained for classification tasks but required to assign continuous scores during inference, resulting in output discrepancies. Manual analysis of the tree generation process reveals that the final layer of Beam Search frequently contains both correct and incorrect candidates that are highly similar, with only subtle distinctions. These minor differences yield comparable scores, making it challenging to determine the optimal solution effectively.

To address this challenge, we propose **BPP-Search**, an algorithm that integrates **B**eam search,

Table 1: Performance evaluation (number of correctly solved examples) of various LLMs on the StructuredOR test dataset (38 examples) under different methods: CoT (Wei et al., 2022) (including CoT-BMLD, where modeling is performed first and then data is imported, and CoT-SPVOC, which follows the sequence of set, parameter, variable, objective, and constraint for CoT), ToT (Yao et al., 2023), SC (Wang et al., 2023), ToT-randomly-chosen (where the final result is obtained randomly from the leaf nodes), ToT-rethink (providing all leaf nodes to the LLM and obtaining a revised result), and ToT-fully-traverse (where every leaf node is thoroughly checked).

Model/Methods	CoT-BMLD	CoT-SPVOC	SC	ToT-Randomly-Chosen	ToT-Rethink	ToT-Fully-Traverse
GPT-4o	**19**	**19**	**21**	**21**	**23**	**30**
GPT-4o-mini	13	14	19	12	19	21
Llama-3-70B	10	17	19	17	17	22
Llama-3.1-70B	18	10	21	5	14	17
Llama-3.2-11B	2	0	1	0	0	8
Qwen-2-72B-Instruct	2	2	4	3	1	5
Qwen-2.5-MATH-72B-Instruct	2	2	0	0	0	4
Qwen-2.5-72B-Instruct	6	5	9	2	2	2
Mixtral-8×7B-v0.1	0	0	0	0	1	1

PRM, and a Pairwise **P**reference model. The core idea is to enhance decision-making in the final layer by leveraging a newly trained **Preference Model**, fine-tuned from Qwen2.5-Math-1.5B, to generate pairwise preference scores for ranking candidates.

Preference Model Data Preparation. To train the preference model for pairwise preferences, we first use the ToT framework to generate a large set of unlabeled reasoning paths and classify them as correct or incorrect based on objective values. For each problem, one correct reasoning path (A) and one incorrect reasoning path (B) are extracted, and all pairwise combinations of correct and incorrect paths are generated. Prompts are constructed by arranging the problem with A and B in two different orders. If A appears first, the prompt is labeled as 1; otherwise, it is labeled as 0. The resulting labeled data is then used to fine-tune the Preference Model as a binary classifier. Details on constructing prompts for the Preference Model are provided in Appendix A.5.

Pairwise Scoring and Candidate Ranking. During inference, the preference model evaluates pairwise preference scores for any two candidates (A, B). The preference score $S_{PM}(A \succ B)$ is calculated as:

$$S_{PM}(A \succ B) = \frac{1}{1 + e^{-l_{pm}}}, \tag{5}$$

where l_{pm} is the logit value for class 1 and $A \succ B$ denotes the preference for A over B.

To compute a comprehensive score for each candidate (A) in the candidate set, pairwise preference scores are aggregated across all other candidates:

$$S_{PM}(A) = \frac{1}{n-1} \sum_{j \neq i}^{j=1,\ldots,n} S(A \succ X_j), \tag{6}$$

where n is the total number of candidates, i is the index corresponding to A, and X_i represents A, while X_j represents other candidates.

Selection of the Optimal Candidate. Using the computed scores $S_{PM}(A)$, candidates are ranked, and the one with the highest score is selected as the optimal answer. This robust ranking mechanism addresses the limitations of the PRM, which struggles to differentiate between similar candidates and accurately identify the correct answer.

Fig. 4 illustrates the BPP-Search process, where nodes are pruned based on PRM scores during beam search (Width = 2), and the Pairwise Preference Algorithm ranks the final candidates to select the optimal solution, ensuring robustness and accuracy.

4.3 Random Greedy Algorithm

To address the limitations of PRM's scoring precision, we employ the **Random Greedy** algorithm. Since PRM provides a rough preference ranking rather than precise scores, randomness is introduced to mitigate the impact of PRM's scoring variability.

The Random Greedy algorithm prioritizes candidates with scores close to the maximum while incorporating randomness to mitigate PRM's imprecision. Candidates are filtered based on the condition:

$$P(a_{\max}) - P(a_i) \leq \text{threshold}, \tag{7}$$

where $P(a_{\max})$ is the highest score, $P(a_i)$ is the score of candidate a_i, and threshold is a predefined margin. From the filtered candidates, one is randomly selected to continue the search process.

Table 2: Number of problems resolved using ToT-Fully-Traverse. The numerator represents the number of examples with at least one correct answer, and the denominator indicates the total number of examples in the test dataset. Results are shown for GPT-4o on the StructuredOR, NL4OPT (Ramamonjison et al., 2022), and ComplexLP (Huang et al., 2024) datasets.

Dataset	Number of Problems Resolved
StructuredOR	30/38
NL4OPT	143/289
ComplexLP	72/211

Table 3: Performance metrics of the PRM on the PRM test dataset (Section 3.3).

Metric	Accuracy	Precision	Recall	F1-Score
Value	0.9823	0.9772	0.9868	0.9820

Table 4: Performance metrics of the Preference Model on the Preference Model test dataset (Section 4.2)

Metric	Accuracy	Precision	Recall	F1-Score
Value	0.7560	0.7761	0.7196	0.7468

5 EXPERIMENT

5.1 Baseline

Given the varying performance levels of policy models across different scales, our objective is to maximize the accuracy of correct results by fully exploring every leaf node in the ToT structure, without requiring fine-tuning of the policy model. Because this approach ensures greater stability and a larger pool of experimental data for subsequent analyses. To achieve this, we design a set of baseline experiments on the **StructuredOR**, providing more reliable and consistent evaluations.

We first evaluate the CoT (Wei et al., 2022) approach, including two variations: CoT-BMLD, where the modeling process is performed first and data is imported later, and CoT-SPVOC, which adheres to the sequence of set, parameter, variable, objective, and constraint in the modeling process. Next, since the ToT (Yao et al., 2023) framework lacks a mechanism to select a final answer from all leaf nodes, we introduce the following configurations to address this limitation: ToT-randomly-chosen, where the final result is randomly selected from the leaf nodes; ToT-rethink, where all leaf nodes are provided to the LLM for reevaluation to produce a revised result; and ToT-fully-traverse, where every leaf node is thoroughly evaluated to ensure that at least one correct result can be generated. The detailed tree structure is shown in the Fig. 2. Additionally, we include SC (Wang et al., 2023) as a baseline, which aims to obtain consistent results by sampling multiple reasoning paths.

The evaluated policy models include GPT-4o (Achiam et al., 2023), GPT-4o-mini (Achiam et al., 2023), Llama-3-70B (Dubey et al., 2024), Llama-3.1-70B (Dubey et al., 2024), Llama-3.2-11B (Dubey et al., 2024), Qwen-2-72B (Yang et al., 2024a), Qwen-2.5-72B (Team, 2024), Qwen-2.5-Math-72B (Yang et al., 2024b), and Mixtral-7×8B (Jiang et al., 2024). Table 1 shows the performance of baseline methods across these models in StructuredOR dataset. The results highlight significant variations in performance, demonstrating how model size and architecture impact their effectiveness in solving problems within the StructuredOR dataset.

To ensure stable performance in subsequent search algorithm experiments across different datasets, we select GPT-4o as the policy model. We first evaluate its ability to solve questions by finding at least one correct answer within the ToT fully-traverse framework on the test datasets from **StructuredOR**, **MAMO-ComplexLP** (Huang et al., 2024), and **NL4OPT** (Ramamonjison et al., 2022). Table 2 presents the results of this evaluation. For subsequent tree search algorithm experiments, only successful cases—where the policy model identifies at least one valid result—are considered. This ensures that the policy model can generate solutions for these questions within the ToT framework.

5.2 Evaluation of BPP-Search

We evaluate our methods on the solvable problems identified in the datasets, as described in Section 5.1. Table 5 presents a comparison between our methods and baseline approaches, focusing on correct rate and reasoning steps. The results show that, under the condition where none of the methods fine-tune the policy model, our methods achieve superior performance with fewer reasoning steps, significantly outperforming baselines.

These experiments validate the feasibility of utilizing PRM to assist inference within the tree-of-thought structure. BPP-Search effectively addresses the limitations of traditional ToT methods, which struggle to reliably select a final result. As shown in Table 5, Greedy Search, Beam Search, and BPP-Search generate better results in significantly fewer steps, exponentially reducing computational costs.

Table 5: Accuracy and reasoning steps for our methods and baselines with a fixed policy model (GPT-4o) on the StructuredOR, Mamo-ComplexLP (Huang et al., 2024), and NL4OPT (Ramamonjison et al., 2022) test datasets. The results are based on 30 problems from StructuredOR, 72 from Mamo-ComplexLP, and 143 from NL4OPT that are confirmed solvable by the policy model in prior experiments.

Method	**StructuredOR**		**Mamo-ComplexLP**		**NL4OPT**	
	Correct Rate	**Steps**	**Correct Rate**	**Steps**	**Correct Rate**	**Steps**
CoT (Wei et al., 2022)	0.633	1	0.486	1	0.566	1
SC (Wang et al., 2023)	0.700	4	0.625	4	0.713	4
ToT-Randomly-Chosen (Yao et al., 2023)	0.700	39	0.444	39	0.629	39
ToT-Rethink (Yao et al., 2023)	0.766	40	0.583	40	0.622	40
Greedy Search Variant (Our Method)	0.833	9	0.555	9	0.713	9
Beam Search Variant (Our Method)	0.800	15	0.666	21	0.783	15
BPP-Search Variant (Our Method)	**0.933**	15	**0.722**	21	**0.804**	15

Table 6: Accuracy and reasoning steps for ablation study of our methods with a fixed policy model (GPT-4o) on the StructuredOR, Mamo-ComplexLP (Huang et al., 2024), and NL4OPT (Ramamonjison et al., 2022) test datasets. The results are based on the same 30 problems from StructuredOR, 72 from Mamo-ComplexLP, and 143 from NL4OPT, confirmed solvable by the policy model in prior experiments.

Method	StructuredOR Correct Rate	Mamo-ComplexLP Correct Rate	NL4OPT Correct Rate	Reasoning Step
Greedy Search + PRM	0.733	0.555	0.699	9
Random Greedy Search + PRM	0.833	0.513	0.692	9
Epsilon Greedy Search + PRM	0.733	0.500	0.713	9
Beam Search (Width=2) + PRM	0.800	0.652	0.783	15
Beam Search (Width=3) + PRM	0.766	0.666	0.755	21
BPP-Search (Width=2)	**0.933**	0.652	**0.804**	15
BPP-Search (Width=3)	0.866	**0.722**	0.797	21

5.3 Ablation Study

Table 6 presents the performance of our methods under different configurations. It is evident that the PRM struggles to assign precise scores for regression tasks. For instance, as the beam search width increases, the accuracy tends to decrease, and in some cases, the performance of beam search becomes comparable to that of greedy search. Manual analysis of the beam search results in the final layer reveals that the candidate queue sometimes contains both correct and incorrect answers that are highly similar in structure, with only subtle differences. This similarity leads to comparable scores, making it challenging for the PRM to reliably distinguish between them.

To address this limitation, we introduce BPP-Search, which incorporates a pairwise preference algorithm. Instead of scoring candidates individually, BPP-Search evaluates all pairwise combinations of candidates within the final queue, comparing each pair and averaging the pairwise preference scores for each candidate. This approach ensures a more robust evaluation by reducing the bias inherent in relying solely on individual scores. The experimental results in Table 6 demonstrate that this method effectively mitigates the risks associated with the imprecise scoring of the PRM, resulting in improved accuracy and robustness. Additionally, based on the performance of PRM, our random greedy search algorithms can at least guarantee performance comparable to standard greedy search, and in some cases, achieve even better results.

6 Conclusion

In this work, we introduce a new operations research dataset that integrates natural language questions with their corresponding detailed modeling processes, addressing the limitations of existing open-source datasets that lack comprehensive annotations of the modeling process. We further propose BPP-Search, an advanced algorithm that combines Beam Search, PRM, and a Pairwise Preference mechanism to enhance the ToT framework. BPP-Search effectively accelerates the reasoning process, improves accuracy, and alleviates the scoring imprecision of PRM, thereby ensuring robust and reliable decision-making. Comprehensive experiments conducted on StructuredOR, NL4OPT, and MAMO-ComplexLP datasets highlight the superiority of BPP-Search. Compared to state-of-the-art approaches, such as CoT, SC, and PRM integrated with Greedy or Beam Search, BPP-Search consistently achieves higher accuracy while requiring fewer reasoning steps, demonstrating its efficacy in addressing complex reasoning tasks in operations research.

7 Limitations

7.1 Trade-offs in ToT Structure: Performance and Computational Cost

In the ToT framework, both increasing tree width and deepening the tree can enhance performance. A wider tree, achieved by increasing the number of child nodes at each layer, provides more exploration paths, thereby improving the likelihood of finding optimal solutions. Similarly, greater depth, achieved by dividing tasks into finer-grained nodes (e.g., separating "set" and "parameter" into distinct layers), not only facilitates more structured and detailed reasoning but also offers additional exploration paths, further increasing the likelihood of identifying optimal solutions.

However, both approaches entail significant computational costs. For example, a tree with a height of 4 and a branching factor of 3 requires 39 LLM queries, whereas increasing the branching factor to 4 raises this to 84 LLM queries, thereby exacerbating computational demands, particularly for long prompts (e.g., 4000 tokens). This creates a trade-off between computational cost and performance. In our study, due to limitations in computational resources, we were unable to construct a tree that is both sufficiently deep and wide to fully explore the solution space.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tobias Achterberg. 2019. What's new in gurobi 9.0. *Webinar Talk url: https://www. gurobi. com/wp-content/uploads/2019/12/Gurobi-90-Overview-Webinar-Slides-1. pdf*, 5(9):97–113.

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*.

Neslihan Özgün Demirel and Hadi Gökçen. 2008. A mixed integer programming model for remanufacturing in reverse logistics environment. *The International Journal of Advanced Manufacturing Technology*, 39:1197–1206.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Bezalel Gavish and Stephen C Graves. 1978. The travelling salesman problem and related problems.

David M Gay. 2015. The ampl modeling language: An aid to formulating and solving optimization problems. In *Numerical Analysis and Optimization: NAO-III, Muscat, Oman, January 2014*, pages 95–116. Springer.

Ralph Grishman and Beth M Sundheim. 1996. Message understanding conference-6: A brief history. In *COLING 1996 volume 1: The 16th international conference on computational linguistics*.

William E Hart, Carl D Laird, Jean-Paul Watson, David L Woodruff, Gabriel A Hackebeil, Bethany L Nicholson, John D Siirola, et al. 2017. *Pyomo-optimization modeling in python*, volume 67. Springer.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2024. Mamo: a mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Bruce P Lowerre and B Raj Reddy. 1976. Harpy, a connected speech recognition system. *The Journal of the Acoustical Society of America*, 59(S1):S97–S97.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Eren Özceylan and Turan Paksoy. 2013. A mixed integer programming model for a closed-loop supply-chain network. *International Journal of Production Research*, 51(3):718–734.

Robert Clay Prim. 1957. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. 2022. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, pages 189–203.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2023. Taskbench: Benchmarking large language models for task automation. *arXiv preprint arXiv:2311.18760*.

Richard S Sutton. 2018. Reinforcement learning: An introduction. *A Bradford Book*.

Zhengyang Tang, Chenyu Huang, Xin Zheng, Shixi Hu, Zizhuo Wang, Dongdong Ge, and Benyou Wang. 2024a. Orlm: Training large language models for optimization modeling. *arXiv preprint arXiv:2405.17743*.

Zhengyang Tang, Chenyu Huang, Xin Zheng, Shixi Hu, Zizhuo Wang, Dongdong Ge, and Benyou Wang. 2024b. Orlm: Training large language models for optimization modeling. *arXiv preprint arXiv:2405.17743*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Pascal Van Hentenryck, Laurent Michel, Laurent Perron, and J-C Régin. 1999. Constraint programming in opl. In *International Conference on Principles and Practice of Declarative Programming*, pages 98–116. Springer.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024a. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *ACL*.

Teng Wang, Zhenqi He, Wing-Yin Yu, Xiaojin Fu, and Xiongwei Han. 2024b. Large language models are good multi-lingual learners: When llms meet cross-lingual prompts. *arXiv preprint arXiv:2409.11056*.

Teng Wang, Wing-Yin Yu, Ruifeng She, Wenhan Yang, Taijie Chen, and Jianping Zhang. 2024c. Leveraging large language models for solving rare mip challenges. *arXiv preprint arXiv:2409.04464*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *ICLR*.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. 2024d. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7309–7319, Miami, Florida, USA. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. 2024. Chain-of-experts: When llms meet complex operations research problems. In *ICLR*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. 2024b. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024c. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate

problem solving with large language models. In *NeurIPS*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: Llm self-training via process reward guided tree search. In *ICLR*.

Yao Zhang, Yuan Hu, Jin Ma, and Zhaohong Bie. 2018. A mixed-integer linear programming approach to security-constrained co-optimization expansion planning of natural gas and electricity transmission systems. *IEEE Transactions on Power Systems*, 33(6):6368–6378.

A Appendix

A.1 Modeling Data Format Specification

Tables 7, 8, 9, 10, 11, 12, 13, 14, 15 define the standardized data format for representing mathematical models. This convention provides a structured and consistent way to label and organize modeling data, ensuring clarity and usability across different tasks and datasets.

A.2 Operation Research Dataset Comparison

The StructuredOR dataset, as illustrated in Figure 5, provides not only the objective value but also the complete modeling process, offering a structured and transparent view of optimization problems. In contrast, as shown in Figure 6, the Mamo-ComplexLP (Huang et al., 2024) and IndustryOR (Tang et al., 2024b) datasets include only the objective value as the label, without detailing the modeling process. This limitation makes it difficult to verify the correctness of the data and prevents the application of reinforcement learning to intermediate steps. Similarly, the NL4OPT (Ramamonjison et al., 2022) dataset lacks both a structured modeling process and clear objective values, further complicating the interpretation and validation of results.

A.3 An Example to Illustrate Reasoning in Mathematical Modeling

Figure 7 illustrates the reasoning steps in the X-of-Thought (XoT) framework under the modeling structure defined in Appendix A.1. The process follows a structured sequence, starting with the question (Q), then progressing through sets (S), parameters (P), variables (V), objectives (O), and constraints (C). Each step builds upon the previous one, progressively transforming the natural language question into a fully defined mathematical model.

A.4 PRM Training Data Collection

To augment positive data for Process Reward Model training, we adopt the following four strategies:

1. **Utilizing ground truth:** Segment the ground truth data into accumulative chunks corresponding to different layers of the reasoning process. This ensures that the hierarchical structure of the data is preserved.

Name	Type	Required	Description
set	list	No	Collection definitions needed for abstract modeling
parameter	list	No	Constants needed for abstract modeling
variable	list	Yes	Variables required for modeling
objective	list	Yes	Objective function definition required for modeling
constraint	list	No	Constraints needed for modeling

Table 7: Summary of components for abstract modeling

Name	Type	Description
name	str	Collection name, must meet programming naming conventions, no spaces allowed
description	str	Description of the collection
data	list	Use a list starting from 1 and ending at the size of the set to represent the number of elements in the set.

Table 8: Details of the Set Component

Name	Type	Description
name	str	Parameter name, must meet programming naming conventions, no spaces allowed
description	str	Parameter description
domain	str	The index dimension of the parameter, e.g., “a <in>Aircraft”. If this parameter is a constant, the domain is an empty string. If this parameter is multi-dimensional, please list the corresponding index.
data	list	Use a list or a number depending on whether the domain is an empty string. If the domain is an empty string, the data is a number. Otherwise, it is a list that can be either one-dimensional or multi-dimensional, representing the values of each parameter across different sets. There is a one-to-one correspondence between the dimensions of data and the domain.

Table 9: Details of the Parameter Component

Name	Type	Description
name	str	Variable name, must meet programming naming conventions, no spaces allowed
description	str	Variable description
domain	str	Index dimension of the variable, e.g., “a <in>Aircraft"
type	str	Variable type: CONTINUOUS, INTEGER, BINARY. Default is CONTINUOUS. Case insensitive

Table 10: Details of the Variable Component

2. **Leveraging LLM-generated data:** Identify correctly generated data from LLMs operating under ToT, CoT, and SC frameworks, and apply the same segmentation operations used for correct generated data. This approach expands the dataset with additional examples while ensuring consistency and alignment with the hierarchical structure.

3. **Swapping indices in summation constraints:** Exchange indices within summation functions in constraints derived from ground truth data. This operation does not alter the fi-

Name	Type	Description
name	str	Objective function name, must meet programming naming conventions, no spaces allowed
description	str	Objective function description
sense	str	Optimization direction of the objective function: min, max, minimize, maximize
function	str	Formula of the objective function, details below

Table 11: Details of the Objective Component

Name	Type	Description
name	str	Constraint name, must meet programming naming conventions, no spaces allowed
description	str	Constraint description
domain	str	Index dimension of the constraint. Without filter: “a <in>Aircraft".
function	str	Formula of the constraint, details below

Table 12: Details of the Constraint Component

Type	Expression	Description
Formula with sum symbol	$\sum_{i \in I} x_i$	$\{i \in I\}$ is the summation dimension

Table 13: Details of Formula Expressions

Type	Expression	Description
Sum dimension with subscript parameter	$\sum_{i \in \text{Successors}_k}$	Does not support Successors_k forms
Nested parentheses in index dimension	$\{i \in P\{k \in A\{l \in A\}\} \in \text{NOE}_i\}$	Nested parentheses in index dimension are not supported
Subscript restriction	$x_{i,j}$ where i, j cannot be a number	Numeric subscripts like $x_{i,1}$, $x_{1,j}$ are not supported

Table 14: Unsupported Formulas and Their Limitations

Type	Expression	Description
Continuous expression	$a < b < c$	Continuous expressions are supported.
Expression separated by commas	$x + y < 0,\ y + z < 1$	Must be separated by English commas; will be split into two constraints.
Two consecutive sums	$\sum_{i \in I} \sum_{j \in J} x_{ij}$	Will be merged into: $\sum_{i \in I, j \in J} x_{ij}$.
Missing * multiplication symbol	$\sum_{i \in I} (a_i x_i + b_i y_i)$	Multiplication symbol will be automatically filled.

Table 15: Supported Special Formulas

nal result, thereby introducing diversity while preserving correctness.

4. **Modifying inequalities:** Swap the left-hand and right-hand sides of inequalities derived

from ground truth data, and adjust the inequality signs accordingly (e.g., ‘>=‘ becomes ‘<=‘). This operation creates valid variations of the data while maintaining correctness.

To augment incorrect data for Process Reward Model training, we apply the following strategies:

1. **Mismatch instance data:** Replace the correct instance data with mismatched values. For example:
    - Modify the value of a parameter so that it no longer corresponds to the data of the set.
    - Delete or add random data to a ‘set‘.
    - Delete a column from a random dimension of a parameter.
    - Reshuffle the data of a random parameter.
2. **Incorrect format:** Generate data using LLMs based on the training dataset, then select examples that cannot be used for modeling due to structural inconsistencies or formatting issues.
3. **Constraint modifications:** Introduce errors in constraints or objectives by:
    - Changing a greater-than sign into a less-than sign.
    - Swapping the indices within a constraint.
    - Altering the summation domain of a constraint.
    - Randomly deleting a constraint.
    - Modifying the function in either the constraint or the objective.
4. **Objective reversals:** Convert a minimization objective into its maximization counterpart, or vice versa.
5. **Generated incorrect models:** Utilize LLM-generated data that is structurally valid and adheres to modeling conventions but produces incorrect results, where the objective value from the modeling solution deviates from the expected outcome. This approach ensures the data aligns with modeling principles while intentionally introducing errors in reasoning or optimization outcomes.

A.5 PRM and Preference Model Prompt

Figure 8 shows two functions illustrating how PRM and Preference Model construct prompts.

Input:

You are managing a large farm in Nebraska and need to determine the most cost-efficient combination of different fertilizers to meet your crop nutrient requirements. The objective is to minimize the total cost of the fertilizer mix. You have a set of available fertilizers, each with a specific cost per unit and nutrient content. Additionally, you need to ensure that the selected fertilizer mix delivers sufficient nutrients essential for healthy crop growth based on soil test outcomes.
The nutrient requirement for your crops is 13 units of nitrogen.
Available fertilizers for your consideration include:
1. Fertilizer A, which costs $17.0 per unit and provides 4 units of nitrogen per unit.
2. Fertilizer B, which costs $19.6 per unit and provides 1 unit of nitrogen per unit.
3. Fertilizer C, which costs $12.2 per unit and provides 2 units of nitrogen per unit.
4. Fertilizer D, which costs $10.6 per unit and provides 2 units of nitrogen per unit.
You need to formulate an optimization problem that minimizes spending while ensuring at least 13 units of nitrogen are available for your crops, with the constraint that the quantity of each fertilizer used must be non-negative.

Output:


```
{
 "objective_value":55.25,
 "set": [
  {
   "name": "Fertilizers",
   "description": "set of available fertilizers",
   "data": [1, 2, 3, 4]
  },
  {
   "name": "Nutrients",
   "description": "set of nutrients required by the crops",
   "data": [1]
  }
 ],
 "parameter": [
  {
   "name": "R",
   "description": "nutrient requirements for the crops",
   "domain": "{n <in> Nutrients}",
   "data": [13]
  },
  {
   "name": "C",
   "description": "nutrient content in each fertilizer",
   "domain": "{f <in> Fertilizers, n <in> Nutrients}",
   "data": [[4], [1], [2], [2]]
  },
  {
   "name": "Cost",
   "description": "cost per unit of each fertilizer",
   "domain": "{f <in> Fertilizers}",
   "data": [17.0, 19.6, 12.2, 10.6]
  }
 ],
 "variable": [
  {
   "name": "Q",
   "description": "quantity of each fertilizer used in the mix",
   "domain": "{f <in> Fertilizers}",
   "type": "continuous"
  }
 ],
 "objective": [
  {
   "name": "TotalCost",
   "description": "minimize total cost of fertilizer mix",
   "sense": "min",
   "function": "<sum_{f <in> Fertilizers} Cost_f * Q_f>"
  }
 ],
 "constraint": [
  {
   "name": "NutrientSufficiency",
   "description": "sufficient nutrients from the mixed fertilizers",
   "domain": "{n <in> Nutrients}",
   "function": "<sum_{f <in> Fertilizers} C_{f,n} * Q_f> >= R_n"
  }
 ]
}
```


Figure 5: An example showcasing a concrete question and its structured modeling process as the label in the StructuredOR dataset.

Mamo-ComplexLP

Input:

Imagine you are in charge of planning a week's worth of meals for a small group and need to ensure everyone gets the right amount of nutrients without overspending. You have four different food items to choose from, each with its own nutritional content and cost.
Here's what each food item offers:
- Food_1: Provides 19 grams of protein, 9 grams of carbohydrates, and 83 calories for $7.
- Food_2: Offers 4 grams of protein, 16 grams of carbohydrates, and 166 calories for $7.
- Food_3: Contains 3 grams of protein, 11 grams of carbohydrates, and 71 calories for $10.
- Food_4: Delivers 8 grams of protein, 7 grams of carbohydrates, and 56 calories for $2.
Your goal is to meet the following nutritional requirements for the group:
- At least 84 grams of protein,
- At least 132 grams of carbohydrates,
- At least 1990 calories.
Your challenge is to determine the most cost-effective way to purchase these food items to meet or exceed the nutritional requirements. What is the minimal cost to meet these dietary needs?

Output:

{"objective_value": "72.0"}

IndustryOR

Input:

The Zhang family has 6 children, Harry, Hermione, Ron, Fred, George, and Ginny. The cost of taking Harry is $1200, Hermione is $1650, Ron is $750, Fred is $800, George is $800, and Ginny is $1500. Which children should the couple take to minimize the total cost of taking the children? They can take a maximum of 4 children on the upcoming trip. Ginny is the youngest, so the Zhang family will take her. If the couple takes Harry, they will not take Fred because Harry doesn't get along with him. If the couple takes Harry, they will not take George because Harry doesn't get along with him. If they take George, they must also take Fred. If they take George, they must also take Hermione. Although this will cost them a lot of money, the Zhang family has decided to take at least three children.

Output:

{"objective_value": "3050.0"}

Figure 6: Comparison highlighting the limitations of the Mamo-ComplexLP and IndustryOR datasets.

Input:

You are managing a large farm in Nebraska and need to determine the most cost-efficient combination of different fertilizers to meet your crop nutrient requirements. The objective is to minimize the total cost of the fertilizer mix. You have a set of available fertilizers, each with a specific cost per unit and nutrient content. Additionally, you need to ensure that the selected fertilizer mix delivers sufficient nutrients essential for healthy crop growth based on soil test outcomes.

The nutrient requirement for your crops is 13 units of nitrogen.

Available fertilizers for your consideration include:

1. Fertilizer A, which costs $17.0 per unit and provides 4 units of nitrogen per unit.
2. Fertilizer B, which costs $19.6 per unit and provides 1 unit of nitrogen per unit.
3. Fertilizer C, which costs $12.2 per unit and provides 2 units of nitrogen per unit.
4. Fertilizer D, which costs $10.6 per unit and provides 2 units of nitrogen per unit.

You need to formulate an optimization problem that minimizes spending while ensuring at least 13 units of nitrogen are available for your crops, with the constraint that the quantity of each fertilizer used must be non-negative.

Reasoning Output:


```
{
"set": [
  {
   "name": "Fertilizers",
   "description": "set of available fertilizers",
   "data": [1, 2, 3, 4]
  },
  {
   "name": "Nutrients",
   "description": "set of nutrients required by the crops",
   "data": [1]
  }
 ],
 "parameter": [
  {
   "name": "R",
   "description": "nutrient requirements for the crops",
   "domain": "{n <in> Nutrients}",
   "data": [13]
  },
  {
   "name": "C",
   "description": "nutrient content in each fertilizer",
   "domain": "{f <in> Fertilizers, n <in> Nutrients}",
   "data": [[4], [1], [2], [2]]
  },
  {
   "name": "Cost",
   "description": "cost per unit of each fertilizer",
   "domain": "{f <in> Fertilizers}",
   "data": [17.0, 19.6, 12.2, 10.6]
  }
 ],
 "variable": [
  {
   "name": "Q",
   "description": "quantity of each fertilizer used in the mix",
   "domain": "{f <in> Fertilizers}",
   "type": "continuous"
  }
 ],
 "objective": [
  {
   "name": "TotalCost",
   "description": "minimize total cost of fertilizer mix",
   "sense": "min",
   "function": "<sum_{f <in> Fertilizers} Cost_f * Q_f>"
  }
 ],
 "constraint": [
  {
   "name": "NutrientSufficiency",
   "description": "sufficient nutrients from the mixed fertilizers",
   "domain": "{n <in> Nutrients}",
   "function": "<sum_{f <in> Fertilizers} C_{f,n} * Q_f> >= R_n"
  }
 ]
}
```


Figure 7: An example illustrating the reasoning process in mathematical modeling.

```python
def build_prm_training_data_template(question, label):
    convention = get_the_convention()

    template = f"""As an expert in mathematical modeling, you are provided with the
conventions for generating instances from concrete problems. The conventions are as follows:

{convention}

Please review the following question:
{question}

And evaluate the following mathematical model (or part of it):
{label}
"""
    return template


def build_dpo_training_data_template(question, model1, model2):
    convention = get_the_convention()

    template = f"""As an expert in mathematical modeling, you are provided with conventions
for generating instances from concrete problems. The conventions are as follows:

{convention}

Please review the following question:
Question: {question}

Evaluate the following two mathematical models:

Model 1 Answer:
{model1}

Model 2 Answer:
{model2}

Instructions:
If Model 1 is correct and Model 2 is incorrect, please consider this example as positive.
If Model 2 is correct and Model 1 is incorrect, please consider this example as negative.
"""
    return template
```

Figure 8: How PRM and the Preference Model construct prompts.