Automated Constraint Model Modification with Large Language Models

Adam Kuca
University of St Andrews, United Kingdom

Phong Le
University of St Andrews, United Kingdom

Nguyen Dang
University of St Andrews, United Kingdom

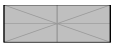
Constraint Programming (CP) is a powerful paradigm for solving combinatorial problems. However, formulating these problems in a constraint modelling language is often challenging and time-consuming, typically requiring experts proficient in CP. To make CP more accessible to non-experts, recent work has explored using Large Language Models (LLMs) for automatic model generation [2]. Yet, generating correct models from natural language remains difficult: current LLMs frequently omit key constraints or produce syntactically invalid code. Moreover, even if an LLM achieves 70%–80% accuracy, the approach may still be impractical for non-expert users, who are unlikely to detect and correct subtle errors in large or complex models. In high-stakes applications, automated constraint modelling must achieve near-perfect accuracy to earn user trust and support real-world adoption.

In this *position* paper, we approach automated modelling from a different angle. Instead of generating models from scratch, we assume that an initial CP model—written by an expert—is already available. We ask: to what extent can LLMs assist in modifying an existing model in response to user-specified changes? This question is highly relevant in practice: real-world deployments are rarely static. Stakeholders often request adjustments, such as adding constraints, relaxing bounds, or changing objectives, necessitating frequent manual edits by developers. Automating this process could significantly reduce expert involvement after deployment and enhance the sustainability of CP-based solutions.

We refer to this new task as *automated constraint model modification*. Intuitively, this task is likely much easier than generating an entire constraint model from scratch, and thus more amenable to achieving near-perfect accuracy with LLMs. In this *ongoing* work, we conduct a preliminary experiment to test this intuition. The remainder of this extended abstract defines the task, outlines a taxonomy of possible modifications, and presents a small-scale evaluation of LLM performance. Our results show that even a zero-shot GPT-4o model produces logically correct edits in 100% of the cases and compiles successfully in 23 out of 24 runs, suggesting that current LLMs may already be reliable for this task.

Task definition. We formulate the task as follows. Given $\mathcal{N}$ the original problem description and its CPMpy model M, we apply an edit $\Delta\mathcal{N}$ to the problem description. The task is to generate the new CPMpy model M' for the modified problem. Modification requests fall into two classes: **parametric** tweaks that only adjust numbers or operators, and **structural** changes that add or remove variables or constraints. We label each edit *easy* (single-line), *moderate* (small multi-line or loop) or *complex* (global, inter-dependent logic). Combining the 8 modification types (from two classes) with three difficulty levels over eight concrete sub-types yields the 24 modification categories.

Dataset. As an ongoing work, we started with the EX5S2 "staircase" problem from [2] (using CPMPy as the modelling language [1]) as the base problem ($\mathcal{N}$). We choose this problem due to its challenge for the original automated constrain model generation task: when using GPT-4o (0-shot) to *generate* the model using the CPMpy pipeline proposed by [2], it produces syntactically correct CPMpy code, yet none of the constraints were correct.



© Author: Please fill in the \Copyright macro;
licensed under Creative Commons License CC-BY 4.0



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

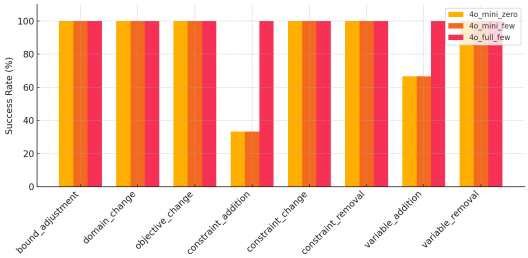


Figure 1 Solution accuracy per modification type for the three models.

For the modification task, we manually crafted 24 edited instances for the EX5S2 example, covering all combinations of (type, difficulty) pairs. Each instance consists of the input $(\mathcal{N}, M, \Delta\mathcal{N})$ and the corresponding ground truth model $M^\star$. We evaluate three settings: `GPT-4o-mini` (0-shot), `GPT-4o-mini` (2-shot), and `GPT-4o` (0-shot). Each prompt includes a system message, the natural language description $\mathcal{N}$ verbatim, followed by clearly delineated blocks titled "### ORIGINAL CODE" and "### CHANGE REQUEST". In the 2-shot setting, the prompt is prepended with two solved examples to guide the LLM. The examples are selected using a static selection strategy; the same two predefined examples are used unless one collides with the current task, in which case a third predefined example is used. Code & data are available at `https://github.com/GSXRcoder159/LLM-CP-Modification`.

Evaluation metrics. Following [2], we report LLM performance using three metrics: (i)*Solution accuracy* (A_{sol}) checks correctness of the solution given by the generated model; (ii) *Constraint accuracy* (A_{cons}) represents the percentage of individual constraints/objective declarations that are logically equivalent to those in $M^\star$; and (iii) *Model accuracy* (A_{model}) checks whether the generated model is logically equivalent to the ground truth.

Results. In Table 1 we present the overall findings, while we break them down by edit type in Figure 1. The most powerful LLM, GPT-4o, solves every parametric edit and all structural edits except one minor syntax issue even without any examples. For the less powerful LLM, GPT-4o-mini, adding 2-shot examples improves the logical constraint accuracy but does not fix the compilation errors.

Model	Runs	Compiles	Cons. acc.	Full equiv.
GPT-4o-mini 0s	24	21 (87.5%)	91.4%	21 (87.5%)
GPT-4o-mini 2s	24	20 (83.3%)	95.2%	21 (87.5%)
GPT-4o 0s	24	**23 (95.8%)**	**99.5%**	**24 (100%)**

Table 1 Aggregate performance on 24 modification examples.

Model equivalence reaches 100% even though constraint accuracy is 99.5% because the latter checks one declaration at a time. The single mismatch arises in a *moderate parametric* modification "allow only step size 5". Our ground truth kept a redundant upper-bound constraint, whereas GPT-4o removed it and thus achieved the same semantics with one constraint fewer—perfect at model level, but scored as a miss at constraint level.

Our small-scale experiment shows that GPT-4o already edits CP models almost flawlessly. The remaining issues are minor syntax typos that can likely be fixed with an iterative loop. Small models needs either better-matched in-context examples, solver-in-the-loop iteration, or some fine-tuning on synthetic edit pairs. In future work, we will (i) grow the benchmark to include more base models and edit combinations, (ii) study more sophisticated retrieval-augmented prompting techniques, and (iii) create a scaffold for iterative in-the-loop fixes of compile or solve errors.

References

1 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.

2 Kostis Michailidis, Dimos Tsouros, and Tias Guns. Constraint Modelling with LLMs Using In-Context Learning. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:27, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2024.20`, `doi:10.4230/LIPIcs.CP.2024.20`.