

Exploiting Large Language Models for the Automated Generation of Constraint Satisfaction Problems

Brown Bag Session, February 6th 2025

originally presented at the
Configuration Workshop 2024
September 3rd, Girona, Spain

will be presented at AAAI 2025

Lothar Hotz¹, Christian Bähnisch¹, Sebastian Lubos², Alexander Felfernig²,
Albert Haag³ and Johannes Twiefel⁴

Hamburger Informatik Technologie-Center, Universität Hamburg, Germany¹

Graz University of Technology, Graz, Austria²

Product Management Haag GmbH, Bad Dürkheim, Germany³

exXa GmbH, Hamburg, Germany⁴

Motivation

- Constraints are solved by developing a constraint program with a constraint tool.
- A constraint program is a software program which uses a certain library for representing constraints.
- LLMs can write software programs.
- Can LLMs write a constraint program for a given constraint problem?



ChatGPT: Du kennst das 8-Dame Problem als Constraint-Problem. Mal ein Bild, welches das 4-Dame-Problem illustriert, also nur 4 Damen auf einem Schachbrett, so aufgestellt, dass sie sich nicht bedrohen.

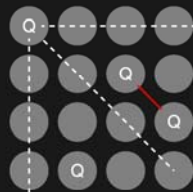
© Lothar Hotz, Universität Hamburg, HITeC e.V.



ChatGPT: Zeige auf einem Schachbrett der Größe 4x4 nur Damen, die sich nicht bedrohen.

© Lothar Hotz, Universität Hamburg, HITeC e.V.

4-QUEENS PROBLEM



Christian Bähnisch, Vortrag HITeC

© Lothar Hotz, Universität Hamburg, HITeC e.V.

8-Queens as Constraint Problem

```

from pycoco.model import Model
from test_tool import verify_solutions
model = Model("8-Queens")
# Create an array of 8 variables representing the positions of queens on a
# chess board
queens = model.intvars(8, 1, 8)
# Add constraints
for i in range(8):
    for j in range(i + 1, 8):
        # Queens cannot be on the same row
        model.arithm(queens[i], "=", queens[j]).post()
        # Queens cannot be on the same diagonal
        model.arithm(queens[i], "-", queens[j], "!=", i - j).post()
        model.arithm(queens[i], "-", queens[j], "!=", j - i).post()
# Find all solutions
solver = model.get_solver()
solutions = []
for solution in solver.find_all_solutions():
    solutions.append([solution.get_int_val( queens[i]) for i in range(8)])
verify_solutions(solutions, "queen8")

```

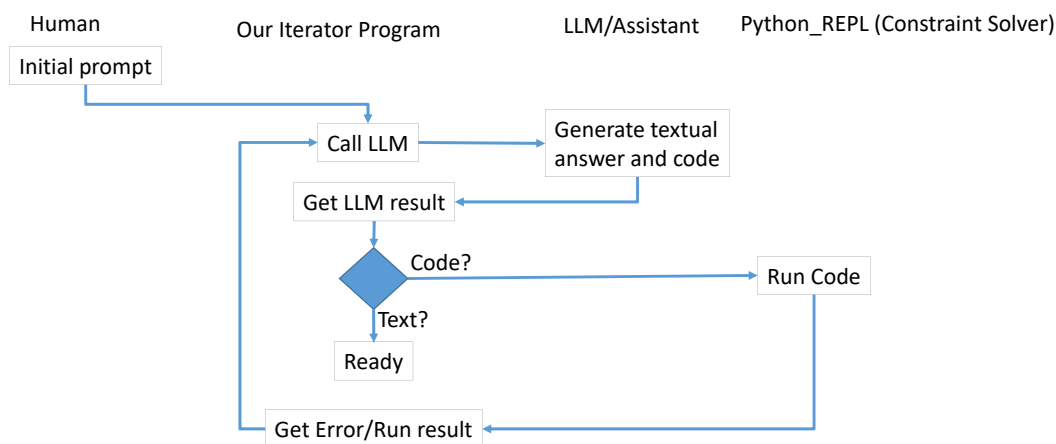
© Lothar Hotz, Universität Hamburg, HITeC e.V.

This Paper

- Solve known constraint problems by generating a constraint program for:
 - 3- and 8-Queens Problem
 - Map Coloring
 - Traveling Salesperson Problem
- How to prompt?
- What to do with syntactically inappropriate, incorrectly generated programs?
- How to check the semantics of the generated program?

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Our Approach

Sep. 3rd 2024, <https://ceur-ws.org/Vol-3812/paper13.pdf>

© Lothar Hotz, Universität Hamburg, HITeC e.V.

The Initial Prompt (1/2)

tool Python_REPL, description: A Python shell. Use this to run and verify your solution to the constraint problem. Input should be a valid Python script using psychoco. The last line should be "verify_solutions(solutions, ...)" in order to verify your solutions

System:

You are an agent designed to solve constraint problems, you write python code to solve and verify constraint problems. You have access to a python REPL, which you can use to execute python code and verify your solution. If you get an error, debug your code and try again. You only use the constraint solver psychoco. You conclude your python code with a variable "solutions" and verify it with the method call "verify_solutions(solutions, <problem_name>)", where <problem_name> is given to you as the first word of the instructions. "solutions" should be a list of all solutions to the constraint problem. If verification fails, you correct your solution.

Use the following example as a template for solving constraint problems with psychoco:

© Lothar Hotz, Universität Hamburg, HITeC e.V.

The Initial Prompt (2/2)

Use the following example as a template for solving constraint problems with psychoco:

User: ints10. find numbers one to 10.

Assistant:

```
''' python
from psychoco.model import Model
from test_tool import verify_solutions
model = Model()
x = model.intvar(1, 10, "x")
solver = model.get_solver()
solutions = [s.get_int_val (x) for s in solver.find_all_solutions
()]
verify_solutions(solutions, "ints10")
'''
```

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Solve Problem Prompts – n-Queen

```
queen3 = "solve the 3-queen problem"
queen8 = "solve the 8-queen problem"
queenn = "solve the n-queen problem, use n=8 as test instance"
```

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Solve Problem Prompts – Map Coloring

```
coloring = """
solve the map coloring problem for four regions , three colors and the given
adjacency :
regions = ['A', 'B', 'C', 'D']
adjacency_list = {
'A': ['B', 'C'],
'B': ['A', 'C', 'D'],
'C': ['A', 'B', 'D'],
'D': ['B', 'C']
}.
the solution should be a list of python dicts where each dict maps regions to
color indices
"""
```

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Solve Problem Prompts – TSP

```
tsp = """ solve the traveling salesman problem , use the following
problem
instance :
# Number of cities
C = 4
# Distance matrix
D = [[0 , 10, 15, 20] , [10 , 0, 35, 25] , [15 , 35, 0, 30] , [20 , 25, 30, 0]]
the solution should be a list of valid solutions , each solution being a list
of
integers representing the cities to be visited .
"""
```

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Information in the Appendix

- Complete dialog: Appendix 1-7
- Errors from Python: Appendix 11
- Number of iterations: Appendix 13

© Lothar Hotz, Universität Hamburg, HITeC e.V.

8-Queens Generated Result

```

from pycoco.model import Model
from test_tool import verify_solutions
model = Model("8-Queens")
# Create an array of 8 variables representing the positions of queens on a
# chess board
queens = model.intvars(8, 1, 8)
# Add constraints
for i in range(8):
    for j in range(i + 1, 8):
        # Queens cannot be on the same row
        model.arithm(queens[i], "!=" , queens [j]).post()
        # Queens cannot be on the same diagonal
        model.arithm(queens[i], "-", queens[j], "!=" , i - j).post ()
        model.arithm(queens[i], "-", queens[j], "!=" , j - i).post ()
# Find all solutions
solver = model.get_solver()
solutions = []
for solution in solver.find_all_solutions():
    solutions.append([solution.get_int_val( queens[i]) for i in range(8)])
verify_solutions(solutions , "queen8")

```

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Observations (1/2)

- For all problems, a constraint code for at least one solution was generated.
- For all problems except TSP, constraint code that computes all solutions was generated.
- The iterative approach which includes syntactic and semantic validation is successful.
- Basic assumption: writing tests for validation is often easier than writing a constraint program, hence, less effort is expected with our approach.
- Not the complete PyChoco API could be provided as LLM context, because of the limited number of tokens

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Observations (2/2)

- The LLM (chatgpt) behaved non-deterministic in our experiments even with temperature set to 0 and fixed seed.
- Similarities to OpenAI Reasoning:
 - Iterative increase of context
 - Shorten of context, if getting too large

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Research Questions (1/2)

- RAG approach with examples of solved constraint problems
 - For textually given mathematical constraint problems
 - For textually given industrial configuration problems
- Can an LLM also solve a constraint problem, not “only” generate a model?
 - Hybrid approach needed?
- Increasing context vs. one-shot context:
 - Can intermediately generated text for the context improve the results?

RAG: Retrieval Augmented Generation

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Research Questions (2/2)

- Generating declarative knowledge with LLMs
 - SAT, PDDL, ASP, Constraints
 - Ontologies
- Context vs. fine-tuned LLMs
 - Train a pre-trained LLM with domain-specific knowledge

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Summary

- Query solutions for known constraint problems
- Iterative result without human in the loop for:
 - Syntactically correct code
 - Semantically validated code
- An LLM can generate constraint programs that solve known constraint problems through our iterative approach.
- Paper: <https://ceur-ws.org/Vol-3812/paper13.pdf>
- Implementation: <https://doi.org/10.5281/zenodo.12794211>
- Next steps:
 - Create constraint programs for new problems

© Lothar Hotz, Universität Hamburg, HITeC e.V.

Thank you for listening!

Contact: lothar.hotz@uni-hamburg.de