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

AAAI 2025 Bridge on
Constraint Programming and Machine Learning
February 26th 2025

Philadelphia, Pennsylvania

Lothar Hotz[1], Christian Bähnisch[1], Sebastian Lubos[2], Alexander Felfernig[2],
Albert Haag[3] and Johannes Twiefel[4]

Hamburger Informatik Technologie-Center, Universität Hamburg, Germany[1]
Graz University of Technology, Graz, Austria[2]
Product Management Haag GmbH, Bad Dürkheim, Germany[3]
exXxa GmbH, Hamburg, Germany[4]

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?

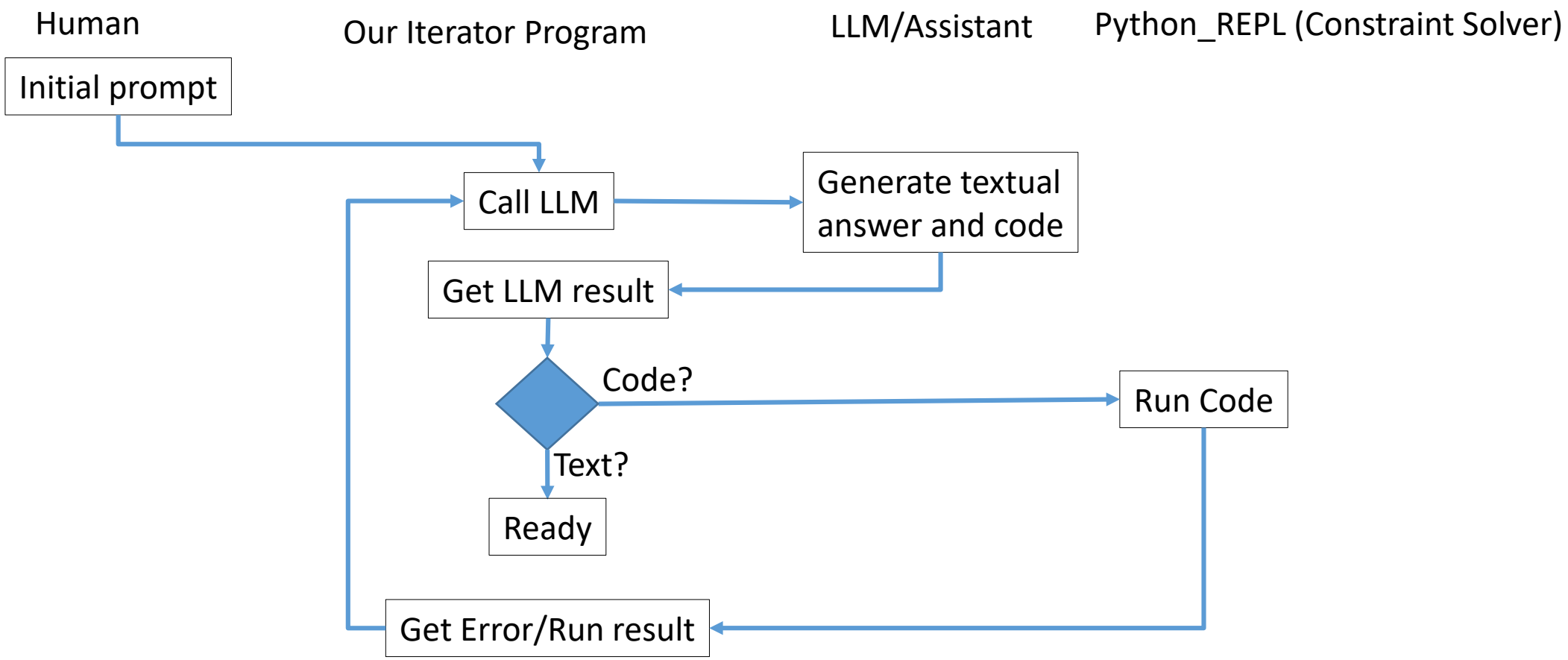
8-Queens as Constraint Problem

```
from pychoco.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")
```

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?

Our Approach



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

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 **pychoco**. 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 pychoco. 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 pychoco:

The Initial Prompt (2/2)

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

User: ints10. find numbers one to 10.

Assistant:

```
‘‘‘ python
from pychoco.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")
‘‘‘
```

Solve Problem Prompts – 8-Queen

queen8 = "solve the 8-queen problem”

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
"""
```

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 .
"""
```

Tool calls

FileName	Tool Calls
coloring_cs_agent_sol2_gpt-4-1106-preview_20240529095402.txt	3
coloring_cs_agent_sol2_gpt-4-1106-preview_20240529095157.txt	1
queen8_cs_agent_sol2_gpt-4-1106-preview_20240529094451.txt	1
queen8_cs_agent_sol2_gpt-4-1106-preview_20240529094710.txt	7
tsp_cs_agent_sol2_gpt-4-1106-preview_20240529090726.txt	15, stopped
tsp_cs_agent_sol2_gpt-4o-2024-05-13-_20240529114219.txt	14, stopped
square_cs_agent_sol2_gpt-4-1106-preview_20240529100358.txt	2
square_cs_agent_sol2_gpt-4-1106-preview_20240529095955.txt	6

See Appendix for all tables

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

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
- Frameworks:
  - https://www.langchain.com/
  - openAI Assistant: https://platform.openai.com/docs/assistants/overview
  - Chain of Thought
  - ReAct: Synergizing Reasoning and Acting in Language Models. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., und Cao, Y. (2023)

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**?

RAG: Retrieval Augmented Generation

Research Questions (2/2)

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

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

Thank you for listening!

Contact: lothar.hotz@uni-hamburg.de

8-Queens Generated Second Last Step

```
from pychoco.model import Model
from test_tool import verify_solutions

model = Model("8-Queens Problem")

# Create an array of integer variables representing the position of each queen.
# Each variable has a domain from 1 to 8, representing the row number in a particular column.
queens = [model.intvar(1, 8, "queen_{}".format(i)) for i in range(8)]

# Constraints: no two queens can be in the same row, column, or diagonal.
model.all_different(queens).post()
for i in range(8):
      for j in range(i+1, 8):
            model.arithm(queens[i], "-", queens[j], "!=", j - i).post()
            model.arithm(queens[i], "+", queens[j], "!=", j - i).post()

# Find all solutions
solver = model.get_solver()
solutions = []
while solver.solve():
      solutions.append([queens[i].get_value() for i in range(8)])

verify_solutions(solutions, "queen8")
```

8-Queens Generated Second Last Step

```
from pychoco.model import Model
from test_tool import verify_solutions

model = Model("8-Queens Problem")

# Create an array of integer variables representing the position of each queen.
# Each variable has a domain from 1 to 8, representing the row number in a particular column.
queens = [model.intvar(1, 8, "queen_{}".format(i)) for i in range(8)]

# Constraints: no two queens can be in the same row, column, or diagonal.
model.all_different(queens).post()
for i in range(8):
      for j in range(i+1, 8):
            model.arithm(queens[i], "-", queens[j], "!=", j - i).post()
            model.arithm(queens[i], "+", queens[j], "!=", j - i).post()

# Find all solutions
solver = model.get_solver()
solutions = []
while solver.solve():
      solutions.append([queens[i].get_value() for i in range(8)])

verify_solutions(solutions, "queen8")
```

Pyton_REPL: 10 of the 746 solutions are correct. Expected 92 solutions
Assistant: It appears that the previous code is not generating all the correct solutions as expected. I will revise the code to ensure that all constraints are correctly implemented and all solutions are found. Let me fix and run the code once more.

8-Queens Generated Last Step - Result

```
from pychoco.model import Model
from test_tool import verify_solutions

model = Model("8-Queens Problem")

# Create an array of integer variables representing the position of each queen.
# Each variable has a domain from 1 to 8, representing the row number in a particular column.
queens = [model.intvar(1, 8, "queen_{}".format(i)) for i in range(8)]

# Constraints: no two queens can be in the same row, column, or diagonal.
model.all_different(queens).post()
for i in range(8):
      for j in range(i+1, 8):
            model.arithm(queens[i], "-", queens[j], "!=", j - i).post() # Different diagonals
            model.arithm(queens[i], "+", queens[j], "!=", i - j).post() # Different diagonals

# Find all solutions
solver = model.get_solver()
solutions = []
while solver.solve():
      solutions.append([queens[i].get_value() for i in range(8)])

verify_solutions(solutions, "queen8")
```

Python_REPL: 92 of the 92 solutions are correct. Expected 92 solutions

Assistant: All 92 solutions to the 8-Queen problem have been found and verified as correct