Test-driven Generation of Constraint Satisfaction Problems Using Large Language Models -Addendum: A Complete Chat for the TSP

Paper #TBD

system:

You are an agent designed to solve constraint problems, you write python code to solve and verify constraint problems. Your code is run and verified by the user. 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: 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") ```

user:

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 vistited.

assistant:

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

Fig. 1. Complete chat for generating a solution for the TSP problem. Part I.

```
C = 4
D = [
 [0, 10, 15, 20],
 [10, 0, 35, 25],
[15, 35, 0, 30],
 [20, 25, 30, 0]
1
model = Model()
The tour: a permutation of cities, starting at city 0 (fix first city)
tour = model.intvars([0] + [1, C-1], C, "tour") # tour[0] = 0
All cities must be visited exactly once
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
To break symmetries, fix the first city to 0
(already done above)
solver = model.get_solver()
solutions = []
for s in solver.find_all_solutions():
 t = [s.get_int_val(tour[i]) for i in range(C)]
 # Make the tour a cycle by returning to 0
 if t[0] == 0 and len(set(t)) == C and all(0 \le city \le C for city in t):
 solutions.append(t)
verify_solutions(solutions, "tsp")
...
```

```
Traceback (most recent call last):
 File "/tmp/tmps5uh3m8t.py", line 15, in <module>
 tour = model.intvars([0] + [1, C-1], C, "tour") # tour[0] = 0
 File "/home/chbaeh/miniconda3/envs/llmconst/lib/python3.10/site-packages/pychoco/
 variables/variable_factory.py", line 71, in intvars
 return [self.intvar(lb, ub, name) for i in range(0, size)]
TypeError: 'list' object cannot be interpreted as an integer
```

### assistant:

```
```python
from pychoco.model import Model
from test_tool import verify_solutions
C = 4
D = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
model = Model()
Fig. 2. Complete chat for generating a solution for the TSP problem. Part II.
```

```
# The tour: a permutation of cities, starting at city 0 (fix first city)
tour = [model.intvar(0, C-1, f"tour_{i}") for i in range(C)]
# All cities must be visited exactly once
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
solver = model.get_solver()
solutions = []
for s in solver.find_all_solutions():
    t = [s.get_int_val(tour[i]) for i in range(C)]
    # Ensure it is a valid tour starting at 0
    if t[0] == 0 and len(set(t)) == C and all(0 <= city < C for city in t):
        solutions.append(t)
verify_solutions(solutions, "tsp")
</pre>
```

2 solutions are correct, 4 are not correct. Expected 2 correct solutions:2 solutions have minimal cost6 solutions are hamiltonian6 solutions start with city 0.

assistant:

```
'''python
from pychoco.model import Model
from test_tool import verify_solutions
C = 4
D = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
1
model = Model()
tour = [model.intvar(0, C-1, f"tour_{i}") for i in range(C)]
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
# Add cost variable
cost = model.intvar(0, 1000, "cost")
# Express the cost of the tour (cycle)
model.sum(
    Γ
        model.element(D, tour[i], tour[(i+1)%C])
        for i in range(C)
    ],
    "=", cost
).post()
```

Fig. 3. Complete chat for generating a solution for the TSP problem. Part III.

```
# Find the minimal cost
solver = model.get_solver()
min_cost = None
for s in solver.find_all_solutions():
    c = s.get_int_val(cost)
    if min_cost is None or c < min_cost:</pre>
        min_cost = c
# Search again, collect only minimal cost tours
model = Model()
tour = [model.intvar(0, C-1, f"tour_{i}") for i in range(C)]
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
cost = model.intvar(0, 1000, "cost")
model.sum(
    [
        model.element(D, tour[i], tour[(i+1)%C])
        for i in range(C)
    ],
    "=", cost
).post()
model.arithm(cost, "=", min_cost).post()
solver = model.get_solver()
solutions = []
for s in solver.find_all_solutions():
    t = [s.get_int_val(tour[i]) for i in range(C)]
    solutions.append(t)
verify_solutions(solutions, "tsp")
...
```

```
Traceback (most recent call last):
    File "/tmp/tmpar_b5jgq.py", line 22, in <module>
       [
       File "/tmp/tmpar_b5jgq.py", line 23, in <listcomp>
       model.element(D, tour[i], tour[(i+1)%C])
       File "/home/chbaeh/miniconda3/envs/llmconst/lib/python3.10/site-packages/pychoco/
       constraints/int_constraint_factory.py", line 245, in element
       if len(table) == 0:
       TypeError: object of type 'IntVar' has no len()
```

assistant:

```
'``python
from pychoco.model import Model
from test_tool import verify_solutions
C = 4
D = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
model = Model()
```

Fig. 4. Complete chat for generating a solution for the TSP problem. Part IV.

```
tour = [model.intvar(0, C-1, f"tour_{i}") for i in range(C)]
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
# Create variables for each arc's cost
arc_costs = [model.intvar(0, max(max(row) for row in D), f"arc_cost_{i}") for i in range
    (C)]
for i in range(C):
    # arc from tour[i] to tour[(i+1)%C]
    # Use element1d to get D[from][to]
    from_var = tour[i]
    to_var = tour[(i+1) & C]
    for from_city in range(C):
        for to_city in range(C):
            # If from_var==from_city and to_var==to_city then arc_costs[i] == D[
                from_city][to_city]
            model.if_then(
                model.and (
                    model.arithm(from_var, "=", from_city),
                    model.arithm(to_var, "=", to_city)
                ),
                model.arithm(arc_costs[i], "=", D[from_city][to_city])
            )
cost = model.intvar(0, 1000, "cost")
model.sum(arc_costs, "=", cost).post()
# First, find the minimal cost
solver = model.get_solver()
min_cost = None
for s in solver.find_all_solutions():
    c = s.get_int_val(cost)
    if min_cost is None or c < min_cost:</pre>
        min_cost = c
# Now, collect all minimal cost tours
model = Model()
tour = [model.intvar(0, C-1, f"tour_{i}") for i in range(C)]
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
arc_costs = [model.intvar(0, max(max(row) for row in D), f"arc_cost_{i}") for i in range
    (C)]
for i in range(C):
   from_var = tour[i]
    to_var = tour[(i+1)%C]
    for from_city in range(C):
        for to_city in range(C):
            model.if_then(
                model.and_(
                    model.arithm(from_var, "=", from_city),
                    model.arithm(to_var, "=", to_city)
                ),
                model.arithm(arc_costs[i], "=", D[from_city][to_city])
            )
cost = model.intvar(0, 1000, "cost")
model.sum(arc_costs, "=", cost).post()
model.arithm(cost, "=", min_cost).post()
                     Fig. 5. Complete chat for generating a solution for the TSP problem. Part V.
```

```
solver = model.get_solver()
solutions = []
for s in solver.find_all_solutions():
    t = [s.get_int_val(tour[i]) for i in range(C)]
    solutions.append(t)
verify_solutions(solutions, "tsp")
```

```
Traceback (most recent call last):
   File "/tmp/tmp8jxdh1xk.py", line 29, in <module>
    model.if_then(
AttributeError: 'Model' object has no attribute 'if_then'
```

assistant:

```
```python
from pychoco.model import Model
from test_tool import verify_solutions
C = 4
D = [
 [0, 10, 15, 20],
 [10, 0, 35, 25],
 [15, 35, 0, 30],
 [20, 25, 30, 0]
1
max_cost = max(max(row) for row in D)
def get_arc_cost_expr(model, from_var, to_var, arc_cost_var):
 # add table constraint for (from_var, to_var, arc_cost_var)
 tuples = []
 for from_city in range(C):
 for to_city in range(C):
 tuples.append([from_city, to_city, D[from_city][to_city]])
 model.table([from_var, to_var, arc_cost_var], tuples).post()
model = Model()
tour = [model.intvar(0, C-1, f"tour_{i}") for i in range(C)]
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
arc_costs = [model.intvar(0, max_cost, f"arc_cost_{i}") for i in range(C)]
for i in range(C):
 get_arc_cost_expr(model, tour[i], tour[(i+1)%C], arc_costs[i])
cost = model.intvar(0, 1000, "cost")
model.sum(arc_costs, "=", cost).post()
```

Fig. 6. Complete chat for generating a solution for the TSP problem. Part VI.

```
First, find the minimal cost
solver = model.get_solver()
min_cost = None
for s in solver.find_all_solutions():
 c = s.get_int_val(cost)
 if min_cost is None or c < min_cost:</pre>
 min_cost = c
Now, collect all minimal cost tours
model = Model()
tour = [model.intvar(0, C-1, f"tour_{i}") for i in range(C)]
model.all_different(tour).post()
model.arithm(tour[0], "=", 0).post()
arc_costs = [model.intvar(0, max_cost, f"arc_cost_{i}") for i in range(C)]
for i in range(C):
 get_arc_cost_expr(model, tour[i], tour[(i+1)%C], arc_costs[i])
cost = model.intvar(0, 1000, "cost")
model.sum(arc_costs, "=", cost).post()
model.arithm(cost, "=", min_cost).post()
solver = model.get_solver()
solutions = []
for s in solver.find_all_solutions():
 t = [s.get_int_val(tour[i]) for i in range(C)]
 solutions.append(t)
verify_solutions(solutions, "tsp")
* * *
```

user: All solutions are correct

Fig. 7. Complete chat for generating a solution for the TSP problem. Part VII.