CSE579: Project Milestone 3 Automated Warehouse Scenario Progress Report

Written by Youmi Koh
Arizona State University
ykoh7@asu.edu

Abstract

This report outlines progress made towards representing the Automated Warehouse Scenario using answer set programming, and motivation behind why certain constraint representations were chosen over others. It highlights challenges with grounding, conflicting preconditions and effects, and the approach taken to address these problems.

Problem Statement

The Automated Warehouse Scenario describes a warehouse environment where a set of robots are tasked with delivering products from inventory stock shelves to designated order picking stations for fulfillment. Orders may include multiple products, and there are various restrictions on to how the robots can operate. The objective is to devise a plan to fulfill all orders in as few steps as possible.

Scenario Description

The scenario starts at time `T=0`. The components of the warehouse scenario can be partitioned into: those that are static and unchanging over time, and others that represent the state of the warehouse at a given time. The latter is termed *fluents*, and can be affected by both time and actions.

Static Objects

Inside the warehouse, the floor is represented in a grid format. Every cell on the grid floor is expressed as **`node(N,(X,Y))`** where **`(X,Y)`** are the cell coordinates, and symbolize a location of the warehouse floor where physical objects can be positioned. Certain cells of the warehouse floor are designated as highways and order picking stations, expressed as **`highway(H,(X,Y))`** and **`pickingStation(PS,(X,Y))`** respectively.

Fluents and Actions

All robots **`R`** and shelves **`S`** are located at a node, and their position at time **`T`** is expressed as **`robotAt(R,(X,Y),T)`** and **`shelfAt(S,(X,Y),T)`** respectively. Products can be stocked in multiple shelves, and inventory levels are expressed as **`inventoryOn(P,S,Q,T)`** where **`Q`** denotes the quantity of product **`P`** stocked on shelf **`S`** at time **`T`**.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Robots are mobile, and can move in 4 cardinal directions. The movement of robot **`R`** at time **`T`** is expressed as **`move(R,(DX,DY),T)`** where **`(DX,DY)`** is a directional unit vector. Robots can pick up and put down shelves, which enables them to carry shelves as they move and relocate the shelf at a new position. The action of a robot picking up a shelf is expressed as **`pickup(R,S,T)`**, and the action of a robot putting down a shelf is expressed as **`putdown(R,S,T)`**. The boolean value of **`B`** in **`carrying(B,R,S,T)`** is set to *true* when robot **`R`** is carrying shelf **`S`**, and *false* otherwise.

The warehouse must fulfill all orders **`O`** at its designated picking station. Orders are composed of multiple line items that state the number of units of a product **`P`**, expressed as **`lineItem(O,P,U,T)`** where **`U`** denotes the remaining number of fulfillable units of product **`P`** at time **`T`**. The action of a robot delivering **`D`** units of a given order's line item is expressed as **`deliver(R,O,P,D,T)`**.

Rules and Constraints

Starting with the domain independent axioms, we use choice rules to set constraints for each fluent to be initially exogenous, inertial, and ensure the existence and uniqueness of their value. Although initially exogenous, when a starting instance is provided, fluents adopt the initial state as defined in each instance. Actions are also declared as exogenous using choice rules. The following is an example of the **`robotAt`** fluent and **`move`** action.

```
{robotAt(R,(X,Y),0)}=1 :- robot(R).
{robotAt(S,(X,Y),T+1)} :- robotAt(S,(X,Y),T), T=0..m-1.
:- not {robotAt(R,(X,Y),T)}=1, robot(R), T=0..m.
robotAt(R,(X,Y),0) :- % initialize with instance state
    init(object(robot,R),value(at,pair(X,Y))).
{move(R,(DX,DY),T)} :-
    robot(R), DX=-1..1, DY=-1..1, T=0..m.
```

The **`move`** action directly affects **`robotAt`** since it changes the location of the robot. This can be expressed recursively as follows:

```
robotAt(R,(X+DX,Y+DY),T+1) :-
    move(R,(DX,DY),T), robotAt(R,(X,Y),T), T=0..m-1.
```

We also declare precondition constraints to restrict the robot from moving diagonally, and disallow moving to a node that is occupied by a robot. Note this covers the case where two robots swap positions, since no robot can move to a node *currently* occupied by another robot, regardless of whether it will move in the same time step. In the move's effect constraints, we ensure that all robots must always be on a valid node (i.e. lands on a valid node post-move) and that no 2 robots can occupy the same node at the same time.

```
:- move(R,(DX,DY),T), |DX|=|DY|, T=0..m-1.
:- move(R,(DX,DY),T), robotAt(R,(X,Y),T),
    robotAt(RR,(X+DX,Y+DY),T), T=0..m-1.
:- robotAt(R,(X,Y),T), not node(_,(X,Y)).
:- 2{robotAt(R,(X,Y),T)}, node(N,(X,Y)), T=0..m.
```

Since shelves are relocated by robots, their location is declared in terms of the robot's location and the **`carrying`** fluent state. Similarly to the robots, we also limit two shelves from occupying the same node at the same time.

```
shelfAt(S,(X+DX,Y+DY),T+1) :-
    move(R,(DX,DY),T), carrying(t,R,S,T),
    robotAt(R,(X,Y),T), shelfAt(S,(X,Y),T), T=0..m-1.
:- 2{shelfAt(S,(X,Y),T)}, node(_,(X,Y)), T=0..m.
```

The **`carrying`** fluent is directly affected by the **`pickup`** and **`putdown`** actions. We add precondition constraints to ensure robots can only pick up shelves occupying the same node, and must not already be carrying any other shelf. The put down action can only be performed if the robot is carrying a shelf, and must not occur on a highway node.

```
carrying(f,R,S,0) :- robot(R), shelf(S).
carrying(t,R,S,T+1) :- pickup(R,S,T).
carrying(f,R,S,T+1) :- putdown(R,S,T), carrying(t,R,S,T).
:- pickup(R,S,T),
    robotAt(R,(X,Y),T), not shelfAt(S,(X,Y),T).
:- pickup(R,S,T), 1{carrying(t,R,SS,T)}.
:- putdown(R,S,T), carrying(f,R,S,T).
:- putdown(R,S,T),
    robotAt(R,(X,Y),T), highway(H,(X,Y)).
```

`inventoryOn` and **`lineItem`** fluents are directly affected by the **`deliver`** action, and declared recursively. Precondition constraints are added to ensure that the robot must be carrying the shelf containing the product, and that the order's delivery is occurring at the designated picking station. To prevent inventory quantity and line item units from falling below 0, we restrict the number of units delivered.

```
inventoryOn(P,S,Q-D,T+1) :-
    deliver(R,O,P,D,T), inventoryOn(P,S,Q,T).
lineItem(O,P,U-D,T+1) :-
    deliver(R,O,P,D,T), lineItem(O,P,U,T).
:- deliver(R,O,P,D,T),
    product(P,S,_), carrying(f,R,S,T).
:- deliver(R,O,P,D,T), order(O,PS),
    robotAt(R,(X,Y),T), not pickingStation(PS,(X,Y)).
:- deliver(R,O,P,U,T),
    inventoryOn(P,S,Q,T), lineItem(O,P,U,T), U>Q.
:- deliver(R,O,P,Q,T), inventoryOn(P,S,Q,T),
    lineItem(O,P,U,T), Q>U.
```

Challenges with Representing the Scenario

Grounding

First, I attempted to represent the scenario with only the fluents and actions introduced in the original description. Although most of the fluents were similarly represented to the previous section, there were no ground terms declared. For example, instead of having a ground term **`robot(R)`** and the fluent **`robotAt(R,(X,Y),T)`**, the entire robot object was declared as **`robot(R,(X,Y),T)`**. As constraints were written, the program became unnecessarily complex. Many unused variables were introduced in the rules, and naively I tried to resolve this by using anonymous variables. It became increasingly difficult to debug and understand the program, and given that clingo attempts to ground all terms, it wasn't obvious to me that the lack of ground terms was the root cause of the problem.

While reviewing the live sessions in preparation for the midterm, the "Methodology of ASP in Clingo" slide highlights the first part as generating a search space of potential solutions, which made me realize that my attempts without ground terms was creating a near unlimited search space (Altunkaya 2023). With this in mind, I factored out the ground terms which helped to simplify all rules and constraints. At one point, the below was how I was declaring the exogenous nature of the **`deliver`** action.

```
{deliver(R,O,P,U,T)} :-
    robot(R,(X,Y),T), order(O,P,U,PS,T),
    pickingStation(PS,(X,Y)), T=1..m-1.
```

The same is now expressed as the below, which leverages the use of local variables to further limit the starting search space.

```
{deliver(R,O,P,D,T): order(O,P,U), D=1..U} :-
    robot(R), T=0..m.
```

Balancing Satisfiability and Optimality

I started with very few necessary rules to generate a reasonable search space, and as constraints were added to remove undesirable behaviour, the program often became unsatisfiable. In these cases, I relaxed the new constraint as much as possible to meet satisfiability then worked with small instances to detect any states in violation of the scenario conditions. Declaring an appropriate objective function to fulfill all orders while minimizing the time steps has also been a challenge.

Conclusion

At this current state, I am testing the program with the sample example in the original scenario description, and comparing my results to the featured optimal model. With an objective function maximizing for number of deliver actions, the program is able to fulfill all orders in less time steps than the featured optimal model. There are likely certain actions and states that violate the original conditions, and I will continue to refine the constraints to ensure that the program is able to generate plans that are both satisfiable and optimal.

References

Altunkaya, A. 2023. Cse 579: Knowledge representation & reasoning - module 4: Practice of answer set programming. Spring 2023. Lecture slides.

Erdem, E.; Lee, J.; and Lierler, Y. 2012. Theory and practice of answer set programming. Tutorial Presentation.