ASU CSE 579 Project Milestone-4
Individual Progress Report

Arda KOCAMAN
Master of Computer Science, Summer 2024
akocaman@asu.edu

Abstract

This report briefly summarize the approach taken for Automated Warehouse Project. The objective of this project is to find minimum steps taken for robots to deliver orders given the environment with applying constraints, using ASP solver clingo.

Problem Statement

This project aims to simulate a simplified version of automated planning operations in a warehouse, focusing on programming robots to transport products to picking stations to fulfill orders efficiently. The primary design objective is to find the minimum total number of time steps required to complete all orders. The warehouse is modeled as a rectangular grid where robots can move a cell at a time horizontally or vertically between adjacent cells. Robots must carry shelves with the required products to the designated picking stations while avoiding collisions and adhering to specific constraints like highway. This problem is well-suited for Answer Set Programming (ASP) because it involves complex constraints and optimization criteria. In ASP, we can organize the problem by first generate a search space of potential solutions by defining possible actions and movements of robots, define new atoms in terms of other atoms to describe the evolving state of the warehouse over time, and filtering out invalid solutions by applying constraints to ensure only feasible outcome (and the most optimal one if we program to do so) are considered. Different techniques like Action Description Languages for planning is used for the solution.

Project Background

The first step to thoroughly understand the project was understanding the landscape for the design process. Warehouse project takes place on a rectangular coordinate which involves inner rectangles (aka. nodes) where other objects are located and moves between. Other objects includes robots that do the action of delivery and transportation, shelfs as the storage units, product, highway as solid constraints for movement. The objects are to be located at nodes, as well as the picking station. There needs to be series of actions which make the transformation of objects between the nodes, like for robots to move, pick up, put down, and deliver. For these objects and actions, I designed a extract.lp program that extracts the initial configuration of objects, first setup the configuration from the instance file, later on with each step store the updated locations. Another program is the environment.lp file, in which I'm defining robot actions and its preconditions like robot-shelf interaction and order delivery fulfillment. Also I'm creating the variables like to_deliver that counts how many products left to deliver and numActions that counts up the number of actions (movement) taken. The clingo program will terminate when there's no more products to left and give me the minimum number of action taken and least number of state to achieve for the most optimum solution.

The design process becomes easy to understand and build when the given example is interpretable. This means the natural language equivalent of initial configuration and aftermath of the actions. Further, the visualization of the state helps to digest. The examples of blocks world problem helped me build a foundation for this project. Having the theoretical knowledge of exogeneity and commonsense law of inertia helped me apply the complete solution for this project.

The Approach Taken for the Solution

First, I designed the extract.lp program for converting the input file schema into a format that is easier to read and implement constraints on. The code define the initial positions and are further developed as they interact with robot actions. Setting the initial state (step=0), the rule

location(object(highway,H),pair(X,Y),0):-init(object(highway,H),value(at,pair(X,Y))). means that when setting up the warehouse configuration, if we know from the initial setup (init) that a highway H is placed at a specific location (X, Y), we will register that highway H is on (X, Y) at the start of the operation (step=0). This rule is followed by another related rule that states we're recognizing this highway as an object at the specified location:
object(highway,H):-init(object(highway,H),value(at,pair(X,Y))).

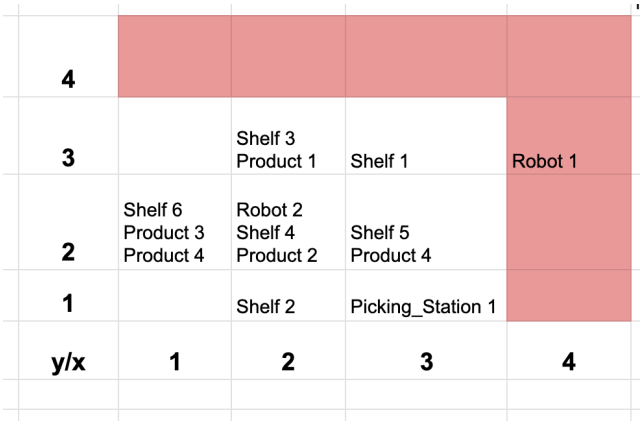
Recognizing objects and mapping them at the given locations allows us to build the configuration at step 0. The state also includes the goal location with pickup locations, enabling us to create the lifeline of the process by mapping both the initial state and the desired state. For orders, the rules such as
object(order,O):-init(object(order,O),value(line,pair(I,U))).
value(line,pair(I,U)):-init(object(order,O),value(line,pair(I,U))). and
object(order,O,value(line,pair(I,U))):-init(object(order,O),value(line,pair(I,U))).
define the initial setup for the orders. Similarly,
object(order,O) :- init(object(order,O),value(pickingStation,P))., object(pickingStation,P) :- init(object(order,O),value(pickingStation,P))., and object(order,O,value(pickingStation,P)) :- init(object(order,O),value(pickingStation,P)). handle the picking stations.

With these definitions, we can start defining the objective variables. I set up these with the to_deliver variable:
to_deliver(C,0) :- C = #count{O,I,U : object(order,O,value(line,pair(I,U)))}. and
:- not to_deliver(0,m). This setup counts the number of product quantities to be delivered initially and ensures that at step m, there are no products left to deliver (count variable C is 0). This defines the start and finish conditions, and the rest of the program finds the most suitable actions that minimize the time steps to reach the end state.

The second part involves setting up the actions for the robots. I defined the movement of the robot with move(1,0;-1,0;0,1;0,-1)., indicating that robots can move only to adjacent nodes (no diagonal movement). Next, I defined the types of actions, the preconditions for those actions, and the effects of those actions. The parsing.asp program processes these initial setups to extract the necessary values, such as the number of robots and shelves. It parses the structured input to set up the initial state of the warehouse environment, ensuring that the data is ready for the main program.

In the main program environment.lp, I defined four types of actions: picking up the shelf, putting it down, delivering products, and moving. Each action has specific conditions and effects. For example, the move action is defined with the rule: occurs(object(robot,R),move(DX,DY),T) :- location(object(robot,R),pair(X,Y),T) , adjacent(pair(X,Y),pair(X+DX,Y+DY)), not location(object(robot,_),pair(X+DX,Y+DY),T). This rule ensures that robots move to adjacent cells and do not switch places with another robot. The pick-up action,
occurs(object(robot,R),pickup,T) :- location(object(robot,R),pair(X,Y),T), location(object(shelf,S),pair(X,Y),T), not carrying(object(robot,R),T)., specifies that robots can pick up shelves if they are in the same cell and not already carrying one. The put-down action,
occurs(object(robot,R),putdown,T) :- carrying(object(robot,R),T), location(object(robot,R),pair(X,Y),T), not location(object(highway,_),pair(X,Y),T)., ensures that robots can put down shelves if they are not on a highway cell. Finally, the deliver action,
occurs(object(robot,R),deliver(O,I,U),T) :- carrying(object(robot,R),T) , order(object(order,O),value(line,pair(I,U))), location(object(pickingStation,P),pair(X,Y),T) , location(object(robot,R),pair(X,Y),T)., ensures that robots deliver the required units of products to the picking stations.

The combined effort of the programs creates a robust system for automating warehouse operations, ensuring efficient and effective order fulfillment within the constraints provided. Interpreting the initial state and input/output rules with connection them to the given example was a good way to test the system. The instance1 program starts with setting up the grid, with the numbers I understand it's a 4x4 grid. Then places objects into those indexes. The second part is optimal plan composed of actions of robot. Visualizing the environment, the action, and again the environment with the effect of the action is helpful. Here's an example of step=0 visualization I've done on inst3.lp:



init(object(order,1),value(pickingStation,1))
means order 1 should be delivered to picking station 1,

and init(object(order,1),value(line,pair(2,1))) means order 1 requires 1 unit of product 2.

The system methodology followed was iterative to better comprehend the problem. Initially, no restrictions were applied, and robots were simulated to move within the given framework with a goal state as a specific node. From there, constraints and actions related to all the system objects were incrementally added. For instance with racks, the pickup and putdown actions of the robots were coded with an end state as a node with a shelf on top of a robot. This iterative approach helped troubleshoot the code and find optimal solutions for each case.

The constraints added during the project were crucial for valid path for shipment. Constraints were created to ensure robots and shelves have unique locations and do not overlap, products of the same type can exist on different shelves, robots can only carry one shelf at a time, and robots are restricted to vertical or horizontal movements only. These constraints ensured that the solution was robust and collision-free. For example, the rule
occurs(object(robot,R),move(DX,DY),T) :- location(object(robot,R),pair(X,Y),T),adjacent(pair(X,Y),pair(X+DX,Y+DY)), not location(object(robot,_),pair(X+DX,Y+DY),T). ensures that robots only move to adjacent cells and avoid collisions.

Results and Analysis

All the given instances ran successfully. In fact, I could do some experimentation by writing very similar instances to try out myself. The results of the project showed that as the number of robots increased, the time taken for deliveries decreased due to simultaneous delivery capabilities. However, as the number of orders increased, the (shortest) delivery time also increased because robots could carry only one shelf at a time. The time taken for the instances were all <1s so I didn't run it with threads. There was no collision observed ensuring the robots reached their goal states in the minimum number of steps. Here's a sample output for instance 5:

This can be interpreted as in instance5's initial conditions the most optimal path requires 10 actions and 6 time steps. The path explicitly tells which robot moves to its adjacent grid, pick up the order and deliver. It's very time consuming trying to visualize all the movements but small checks like starting and end point validity can be done. Again from this instance, the last moves:
occurs(object(robot,1),deliver(1,1,1),4) occurs(object(robot,2),deliver(1,3,4),6)
is correct for the desired delivery location given in the program:
init(object(order,1),value(line,pair(1,1))).
init(object(order,1),value(line,pair(3,4))).
Other checks like collision can also be done occurring same location of different robots at the same time step but this is time consuming.

Opportunities for Future Work

The project taught me a lot particularly the practical application of ASP. The iterative approach and regular team meetings were crucial for debugging and refining the solution. The script ultimately solved all given instances successfully with the minimum number of steps, ensuring the optimality of the solution. This approach can be extended to real-world scenarios, such as Amazon warehouses, by integrating autonomous agents and enhancing scalability to handle larger warehouses and more complex order scenarios. The first thing on my mind is to assign employees instead of robots and have their own constraints.
Although I believe the intuition of this project might exceed the scope of warehouse. Modelling a planning problem by writing logic programs, then find stable models that can be interpreted as solution for the problem approach might be applicable various similar problems like scheduling.

Conclusion

This project taught me a lot about solving real-world challenges with ASP. Since the beginning of my CS education and continuing with professional experience, I always wrote programs in imperative programming languages. This is my first experience developing solutions with declarative programming and constructing the problem search space and do not explicitly control the flow of the solution gave me different perspective into how we could approach the problem itself. The hardest part was to set out the constraints of motions. Even though some constraints seem redundant like shelf cannot be on highway, it still needs to be tediously written as constraint. My approach for incrementally adding the constraints made this challenge a lot easier. In order to figure out the edge conditions, testing with the given scenarios and visualizing moves and their effects helped me.

References

Answer Set Programming (Draft), Vladimir Lifschitz ,University of Texas at Austin. April 5, 2019

Automated Warehouse Scenario Progress Report, Arda Kocaman. June 22, 2024.

A User's Guide to gringo, clasp, clingo, and iclingo, Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Thiele, S. The University of Texas at Austin. October 4, 2010.