An Exploration in Answer Set Programming Applied to Autonomous Warehouses

Trey Manuszak
Arizona State University
Tempe, Arizona
tmanusza@asu.edu

Abstract

Automated warehouses provide many benefits to managing one's supply chain. Those are including, but not limited to, employee safety, minimal product mishandling and displacement, increased traceability and auditability, and increased productivity. Coordinating this automation requires a program that can quickly determine optimal, or as close to optimal as possible, scheduling of the warehouse robotics. In this work, we collect requirements and limitations of an autonomous warehouse, develop a scheduling program using answer set programming, while implementing techniques from the knowledge representation and reasoning field of artificial intelligence to generate a time-optimized course of action for the delivery of product.

Problem Statement

Finding an opimal schedule of robots delivering products in an automated warehouse is known to be a problem in the $\mathcal{NP}$-hard complexity class (Lenstra, Rinnooy Kan, and Brucker 1977). There have been many attempts to solve this problem (1999; 2012a; 2012b; 2016). Answer set programming (ASP) is designed to solve $\mathcal{NP}$-hard search problems. Of the many different ASP languages, we chose to work with `clingo` to build an automated warehouse scheduler to deliver product to picking stations (Gebser et al. 2019).

To develop an automated warehouse scheduling program we must define the constraints of the warehouse, warehouse robotics, products, shelving, and orders, then generate a time optimized route for the robots to deliver products to the picking stations. The constraints we use come from Google's Answer Set Programming Challenge 2019 and can be generally summed up as follows (Google 2019):

- The warehouse consists of tiles, some of which are highway tiles which have their own certain limitations,
- A robot can only move vertically or horizontally, they can move under shelves if they are not carrying one already, they can set shelves down on normal tiles, and they can place product in picking stations that they are next to,
- Products must be on a shelf or at a picking station,
- Shelves may only be placed on normal tiles,
- Orders consist of items to be placed at a specific picking station,
- A picking station must be on a normal tile and cannot be moved.

The program, when given a state of the warehouse and the orders to be completed, generates a schedule for completing the orders in as little time as possible. An order that is assigned to a picking station is completed, if all items in the order are delivered to that picking station. The schedule must correctly complete all orders. The program should find a schedule completing all orders in T steps, where no schedule is possible in less than T steps. It is important to note that the constraints on movement of the robots is not serializable. That is, we are allowed to move a robot R_1, to a space just previously occupied by robot R_2, as long as robot R_2 is also moving to a different space than just previously occupied by robot R_1.

Project Background

The project focuses on applying techniques from the knowledge reasoning and representation subfield of artificial intelligence. In particular, this project requires writing a transition system in answer set programming. To do this, we characterize the states of an automated warehouse and the actions taken inside.

Specifically, the states of the warehouse involve the location of robots, shelving, product, orders, and the layout of the highway tiles, and picking stations. These states have constraints tied to them, which are those generally listed in the previous section. Then, the actions which occur in the warehouse is the movement of robots, picking up and putting down shelving, and delivering product from shelves.

Approach

The following development life-cycle was generally done as follows. We choose an object and action in the warehouse to develop next, and attempted a first implementation. This involves writing the constraints of the state, developing the exogenous actions, defining the direct effect of the actions, writing the uniqueness and existence of state, and

finally adding a law of inertia for a state base case. Then, testing was often done in two parts. First, we would reduce the warehouse to the lowest complexity we were developing for and check for correctness of all possible actions in one and two time-steps. After verifying correctness and fixing patches if necessary, we would increase the complexity of the warehouse and test for unsatisfiability of states that should not be possible from the defined constraints. After testing and patching that, we could be reasonably sure that our implementation was correct.

Initial State

The first action of development was to correctly create the initial provided state of the warehouse. From the provided initial state, we instantiate several facts, such as nodes, highways, picking stations, robots, shelving, products, and orders. The correctness was verified manually against the 5 provided test cases for the project. Objects were added to this section in later development cycles as needed.

Robot Movement

Following the initialization phase, we first developed the robot movement. To show how in more detail how the above testing strategy was performed, the following outlines the testing for robot movement. We first developed what we thought were the correct constraints and actions of defined movement and tested the implementation on a 1x2 warehouse grid with no shelving, picking stations, product, orders, or highways, and with only one robot. Then, we would test all possible states after one and two time-steps. Then, we would test for unsatisfiability of states that should break the defined constraints. For example, we would test if moving off the defined warehouse grid was satisfiable. Once we verified the correctness of a one robot, limited complexity warehouse, we then tested in those two manners above in a more complex environment, such as multiple robots in a larger grid.

Shelving Actions

After achieving correct robot movement, we then moved on to shelving. After writing shelving placement constraints, such as one shelf per node and maximum one robot per shelf, we tested for correct instantiation of shelving. We also noted to develop uniqueness and existence properties of shelving locations. Since shelves themselves do not perform any action, we could not do the two step testing process provided above, until picking up and putting down shelving was developed.

We then developed the required code for robots picking up shelving. This also required an additional three argument "carry" state that denoted a robot R carrying a shelf S at time-step t. The development cycle for this part was thought to be correct after the testing phase until testing for putting down actions were being tested. We had to return to this code later and add a statement noting that if a robot R was carrying a shelf at time-step $t-1$ and did not put it down at time-step t, then it should still be carrying the shelf at time-step t.

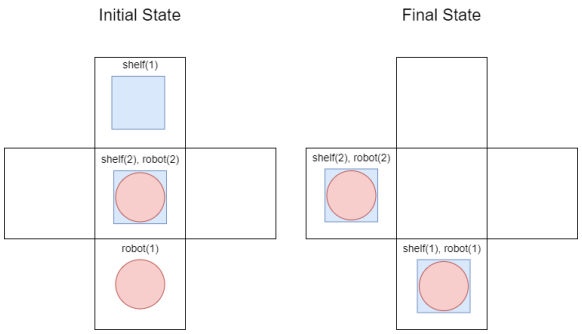

Figure 1: Higher complexity testing scenario for shelving actions and constraints.

After the initial development of the pickup and put down actions, the higher complexity testing involved an initial warehouse state in the shape of a cross where there are two robots. One robot is in the intersection of the two hallways and the other is at one end of one of the hallways. There were also two shelves. One also at the intersection, and the other at the opposite end of the hallway. The goal of the test was to have the robot at the end of the hallway carry the shelf at the other end back to its starting location. The time-optimized solution would involve the second robot picking up the shelf at its intersection and moving it out of the way for it's coworker. A successful test of this could reasonably give us a high confidence of a correct implementation, as it involves a highly complex strategy of robot cooperation. The test case can be visualized in Figure 1.

Delivering Product

Next, we developed the constraints and actions for the delivery of product from shelves to the order's assigned picking station. This took the most time to develop as it requires a large number of constraints to the state. Specifically we need to define the non-negativity of product on shelves and remaining product in an order. Additionally, we needed both the order-product-unit and product-shelf-unit triple to be in persistent existence and unique. Further, although there is only one action to implement, that is deliver, there are many changes to state to take into account for such an occurrence of a delivery action. These state changes are a reduction in the quantity of product on the shelf as well as a reduction in the quantity of items left in the order. On top of this, we require law of inertia rules for orders and products on shelves.

To implement the above constraints and rules, it was necessary for us to revisit the first development cycle. We needed not only state to address a unique order-product pair, but also state for a specific order-product pair. Similarly we require the additional state for unique and general product-shelf pairs. This can be seen at the end of the delivery code section in Appendix A.

During low complexity testing, We recognized that we didn't implement the above rules to allow a robot to deliver any quantity of product between 1 and the minimum of the units left in the order or existing product on the shelf. It was always just the minimum of the units left in the order or ex-

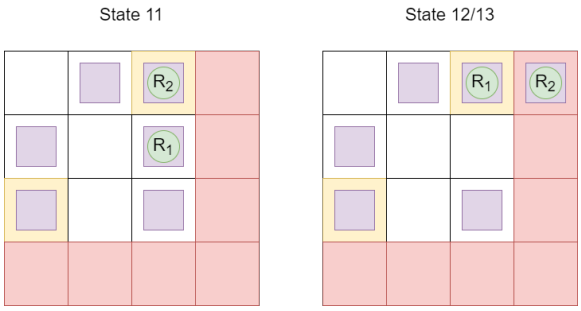

Figure 2: This is the final step(s) for the optimal strategy for provided instance 1. The final step involves robot R_2 moving to the right and robot R_1 moving up during time-step 12 so that R_1 can deliver the final product of the last order to the picking station in yellow. The red tiles are highways, and the purple boxes are shelves. Both robots are carrying the shelves above them during state 11.

isting product on the shelf. It could be possible that an optimal solution may only deliver a less than maximum allowed product in a time-step.

The last step was to implement an optimization parameter for our solution and an end goal of the program. A time-optimal strategy is defined in the project to be one that completes all orders in the smallest time-step. Taking this requirement, we can develop the minimization in two lines of code, seen at the bottom of Appendix A. First, we get the maximum time-step in the strategy, then we minimize the program with respect to this maximum time-step. Lastly, a strategy is complete if it delivers all units in all orders by the largest time-step in the strategy. This can be seen in the last line of Appendix A.

Analysis and Results

Testing that our implementation was correct against the one provided solution to the first sample instance took quite a debugging effort. At first, letting our program find the optimal solution returned an optimum strategy in 15 steps. The true optimal strategy should have been 13 steps. So there was a bug in our program that needed to be tracked down. To do this, we programmed the actions that should have been taken at the end of each time step. Starting at time step 1, we checked to see if each time-step were satisfiable with our program. They were all satisfiable all the way up to step 11 of 13. You can see the optimal states of the warehouse for this instance at state 11 and 12/13 in Figure 2. At first our thought was that it was a serializability issue, as the last required movement is not serializable, which is defined to be acceptable. We checked our program and confirmed that we did constrain our program to be serializable.

Our next strategy for tracking down the bug was to list all of the actions our program allowed from state 11 to state 12. There were 7 returned to us in total. Those were a combination of robot R_1 moving to the left or setting down it's shelf and robot R_2 moving down or setting down its shelf, or the default do nothing action set. It from our returned actions that our implementation wasn't allowing the robots to travel on the highway cells even if they were carrying a shelf, which is allowed. After revisiting the rules that were culprit to the bug and patching them, we were able to correctly achieve the optimal solution of 13 steps for instance 1.

Instance	1	2	3	4	5
Robots	2	2	2	2	2
Shelves	6	5	6	6	6
Picking Stations	2	2	1	2	1
Product Types	4	3	4	2	4
Orders	3	2	2	3	1
Optimal Strategy Length	13	11	9	10	10
Time to Compute (sec)	5.5	3.3	1.7	3.0	1.6

Table 1: This is for general reference of the 5 provided initial instances along with the optimal strategy length and the time required to compute.

Although we won't describe the detailed initial states of the provided instance here, they can be obtained from the author. We refer to Table 1 for a general idea of the size of the instance and for the results of the strategy. All instances are in a 4x4 warehouse grid, with similar node and highway layout to the one shown in Figure 2. The instances were computed on an Intel Core i7-7700K CPU at 4.2GHz with 32 GB of memory.

Future Work

There is still a lot to be completed before the program could be used in a real setting at modern warehouse and supply chain scale and complexities. We can see that with our quite simple instances, we could only classify optimal strategies about 15 steps ahead within 5 seconds on student accessible hardware. As of 2016, Amazon managed approximately 30,000 of it's Kiva robots (Bogue 2016). Now, Amazon maintains over 500,000 Kiva robots to deliver products in its warehouses (Amazon 2022). Managing a number of robots this large would require a program designed towards a distributed computation setting. On our commodity hardware it was difficult to generate strategies even relatively near to the present. Taking even seconds to generate a global strategy for the warehouse that could be acted upon and completed in seconds is not adequate at scale. Attempting such long predicted strategies at an Amazon level scale would require the use of statistics to generate locally optimal strategies rather than global.

However, supply chains require many more actors to manage than just warehouse robots. Taken from the 2021 Amazon Annual Report, they have "253 fulfillment centers, 110 sortation centers, and 467 delivery stations in North America, with an additional 157 fulfillment centers, 58 sortation centers, and 588 delivery stations across the globe. Our delivery network grew to more than 260,000 drivers worldwide, and our Amazon Air cargo fleet has more than 100 aircraft," (Amazon 2021). Not only would an automated warehouse program have to manage robots, but it would be ideal if it could manage and direct beyond the borders of a warehouse to deliver optimal strategies for its manual force to

bring product to its next stage in an efficient and safe manner.

Further, as techniques automated warehouse scheduling improves, so should the architecture of the supply chain. Everything from the physical design and placement of warehouses, trucks, shelves, and product should be explored to better optimize the delivery of goods to customers. Even the invention of new transportation, that can be scheduled by an automated warehouse scheduler, can advance the speed and scale of the supply chain. For example, Amazon uses drones for transporting goods to customers, but utilizing it in the warehouse could also be a way to the increase speed and efficiency of deliveries (Vempati et al. 2017).

Lastly, we need further focus in the future to bring more predictability and resiliency to our supply chains. The development of just-in-time manufacturing hyper-optimized our global supply chains, which caused great delays and fractured transportation routes during the COVID-19 pandemic (Pujawan and Bah 2021). Developing rules and constraints to improves the resiliency of the schedules is necessary today. Although a schedule could be optimal, if just one misplaced item or fallen shelf causes mayhem in your warehouse, that schedule should not be recommended. For a resilient warehouse, our program should be providing schedules that are not outliers, but contain many close schedule neighbors, that are similarly optimal. Then, to improve the predictability of our warehouse deliveries, we should provide schedules that decrease the variance in order completion. Just as in a CPU scheduler, we should consider implementing a priority level to orders and changing this parameter as the order continues to be unfulfilled.

References

Amazon. 2021. Amazon, 2021 annual report. https://s2.q4cdn.com/299287126/files/doc_financials/2022/ar/Amazon-2021-Annual-Report.pdf.

Amazon. 2022. Look back on 10 years of amazon robotics.

Atieh, A. M.; Kaylani, H.; Al-Abdallat, Y.; Qaderi, A.; Ghoul, L.; Jaradat, L.; and Hdairis, I. 2016. Performance improvement of inventory management system processes by an automated warehouse management system. *Procedia Cirp* 41:568–572.

Basile, F.; Chiacchio, P.; and Coppola, J. 2012a. A hybrid model of complex automated warehouse systems—part i: Modeling and simulation. *IEEE Transactions on Automation Science and Engineering* 9(4):640–653.

Basile, F.; Chiacchio, P.; and Coppola, J. 2012b. A hybrid model of complex automated warehouse systems—part ii: Analysis and experimental results. *IEEE transactions on automation science and engineering* 9(4):654–668.

Bogue, R. 2016. Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot: An International Journal* 43(6):583–587.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming* 19(1):27–82.

Google. 2019. Answer set programming challenge 2019. https://sites.google.com/view/aspcomp2019/.

Lenstra, J.; Rinnooy Kan, A.; and Brucker, P. 1977. Complexity of machine scheduling problems. In Hammer, P.; Johnson, E.; Korte, B.; and Nemhauser, G., eds., *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*. Elsevier. 343–362.

Pujawan, I. N., and Bah, A. U. 2021. Supply chains under COVID-19 disruptions: literature review and research agenda. *Supply Chain Forum: An International Journal* 23(1):81–95.

van den Berg, J. 1999. A literature survey on planning and control of warehousing systems. *IIE Transactions* 31(8):751–762.

Vempati, L.; Crapanzano, R.; Woodyard, C.; and Trunkhill, C. 2017. Linear program and simulation model for aerial package delivery: A case study of amazon prime air in phoenix, AZ. In *17th AIAA Aviation Technology, Integration, and Operations Conference*. American Institute of Aeronautics and Astronautics.

Appendix A

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% automated-warehouse.asp %
%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% OBJECTS AND LOCATIONS %%%%
% Here we define objects and object location (if it moves)

% node (cell in warehouse)
node(N,X,Y) :-
init(object(node,N), value(at, pair(X,Y))).

% highway (fast cell in warehouse)
highway(H,X,Y) :-
init(object(highway,H), value(at, pair(X,Y))).

% picking station
pickStation(P,X,Y) :-
init(object(pickingStation, P), value(at, pair(X,Y))).

% robot and robot's location
robot(R) :-
init(object(robot, R), value(at, pair(X,Y))).
robotLocation(R,X,Y,0) :-
init(object(robot,R), value(at, pair(X,Y))),
  node(N,X,Y).

% shelf
shelf(S) :-
init(object(shelf, S), value(at, pair(X,Y))).
shelfLocation(S,X,Y,0) :-
init(object(shelf, S), value(at, pair(X,Y))),
  node(N,X,Y).

% products
productShelfPair(P,S) :-
init(object(product,P), value(on, pair(S,U))).
productOnShelf(P,S,U,0) :-
init(object(product, P), value(on, pair(S,U))).

% order
orderProductPair(O,P) :-
init(object(order, O), value(line, pair(P,U))).
orderDetails(O,P,U,0) :-
init(object(order, O), value(line, pair(P,U))).
deliverTo(O,P) :-
init(object(order, O), value(pickingStation,P)).

%%%% MOVING %%%%
% move options
move(1,0; -1,0; 0,1; 0,-1).

% must be in the warehouse
:- robotLocation(R,X,Y,T),
not node(_, X, Y).

% one robot per node
:- robotLocation(R1,X,Y,T),
robotLocation(R2,X,Y,T), R1!=R2.

% robot movement is exogenous
{occurs(object(robot,R), move(X,Y), T)} :-
robot(R),
move(X,Y),
T=1..m.

% direct effect
robotLocation(R,X + X1,Y + Y1,T) :-
occurs(object(robot,R), move(X1,Y1), T),
  robotLocation(R,X,Y,T-1),
T=1..m.

% uniqueness and existence
:- not {robotLocation(R,X,Y,T):node(N,X,Y)}=1,
robot(R),
T=1..m.

% robots cant switch nodes
% (note, serializability is not required)
:- robotLocation(R1,X1,Y1,T),
robotLocation(R2,X2,Y2,T),
  robotLocation(R1,X2,Y2,T-1),
robotLocation(R2,X1,Y1,T-1), R1!=R2.

%%%% SHELVING %%%%
% shelves cant be on highways, unless carried
:- shelfLocation(S,X,Y,T),
highway(H,X,Y),
not carry(R,S,T),
robotLocation(R,X,Y,T),
T=1..m.

% shelf cant be on two different nodes
:- 2{shelfLocation(S,X,Y,T):node(N,X,Y)},
shelf(S),
T=1..m.

% No two shelves on one node
:- 2{shelfLocation(S,X,Y,T):shelf(S)},
node(N,X,Y),
T=1..m.

% no shelf carried by two different robots
:- 2{carry(R,S,T):robot(R)},
shelf(S),
T=1..m.

% direct effect of carried movement
shelfLocation(S,X+DX,Y+DY,T) :-
robotLocation(R,X,Y,T-1),
carry(R,S,T-1),
occurs(object(robot,R),
move(DX,DY), T),
T=1..m.

% If carrying and didnt put down, still carry
:- carry(R,S,T-1),
not carry(R,S,T),
not putdown(R,S,T),
T=1..m.

% uniqueness and existence of shelf location
:- not {shelfLocation(S,X,Y,T):node(N,X,Y)}=1,
shelf(S),
T=1..m.

%%%% PICKUP %%%%
% pickup is exogenous
{pickup(R,S,T):shelf(S)}1 :-
```

```
robot(R),
T=1..m.
occurs(object(robot,R), pickup, T) :-
pickup(R,S,T).

% Cant pickup from 2 different robots
:- 2{pickup(R,S,T):robot(R)},
shelf(S),
T=1..m.

% cant pickup a shelf, if you already have one
:- pickup(R,S,T),
carry(R,S,T-1),
T=1..m.

% Robot can only pickup if in the same location as the shelf
:- pickup(R,S,T),
shelfLocation(S,X1,Y1,T),
robotLocation(R,X2,Y2,T),
node(N1,X1,Y1),
node(N2,X2,Y2),
N1!=N2,
T=1..m.

% if the shelf is picked up, then it is being carried
carry(R,S,T) :-
pickup(R,S,T),
T=1..m.

%%%% PUTDOWN %%%%
% putdown is exogenous
{putdown(R,S,T):shelf(S)}1 :-
robot(R),
T=1..m.
occurs(object(robot,R), putdown, T) :-
putdown(R,S,T).

% cant be put down by 2 different robots
:- 2{putdown(R,S,T):robot(R)},
shelf(S),
T=1..m.

% Can only put down a shelf if it has one
:- putdown(R,S,T),
not carry(R,S,T-1),
T=1..m.

% if the shelf is put down, then it is not being carried.
not carry(R,S,T) :-
putdown(R,S,T),
T=1..m.

%%%% DELIVERING %%%%
% Exogenous delivery action
{deliver(R, S, O, P, 1..U2, T)}1 :-
robotLocation(R,X,Y,T-1),
pickStation(PS, X, Y),
deliverTo(O,PS),
orderDetails(O, P, U1, T-1),
productOnShelf(P, S, U2, T-1),
carry(R, S, T-1),
T=1..m.
occurs(object(robot,R), deliver(O,P,U), T) :-
deliver(R,S,O,P,U,T).

% product in the order reduces by amount delivered
orderDetails(O,P,U1 - U, T) :-
orderDetails(O, P, U1, T-1),
occurs(object(robot,R),
deliver(O,P,U), T),
T=1..m.

% Amount of product on the shelf reduces by amount delivered.
productOnShelf(P,S,U1 - U, T) :-
productOnShelf(P, S, U1, T-1),
deliver(R,S,O,P,U,T),
T=1..m.

% The reduced quantities may not be negative
:- productOnShelf(P,S,U,T),
U < 0.
:- orderDetails(O,P,U,T),
U < 0.

% Amount delivered needs to be nonzero
:- occurs(object(robot,R),
deliver(O,P,U), T),
U < 1.

% uniqueness and existence of ordersDetails
:- not {orderDetails(O,P,U,T)}=1,
orderProductPair(O, P),
T=1..m.

% uniqueness and existence of productOnShelf
:- not {productOnShelf(P,S,U,T)}=1,
productShelfPair(P, S),
T=1..m.

%%%% LAWS OF INERTIA %%%%
{robotLocation(R,X,Y,T)} :-
robotLocation(R,X,Y,T-1),
T=1..m.
{carry(R,S,T)} :-
carry(R,S,T-1),
T=1..m.
{shelfLocation(S,X,Y,T)} :-
shelfLocation(S,X,Y,T-1),
T=1..m.
{productOnShelf(P,S,U,T)} :-
productOnShelf(P,S,U,T-1),
T=1..m.
{orderDetails(O,P,U,T)} :-
orderDetails(O,P,U,T-1),
T=1..m.

%%%% AXIOMS %%%%
% robot can only do one thing at a time
:- occurs(object(robot, R), A1, T),
occurs(object(robot, R), A2, T),
A1!=A2.

%%%% MINIMIZATION AND GOAL %%%%
max_t(X) :- X =  #max{T:occurs(_,_,T)}.
#minimize{T : max_t(T)}.
:- orderDetails(O,P,_,m),
not orderDetails(O,P,0,m).
```