Automated Warehouse Scenario

Trayson Kelii
Arizona State University

Abstract

The purpose of this paper is to report the results, methodology, and solution for the Automated Warehouse Scenario (AWS). The subject matter will be partitioned into the following sections, namely, Problem Statement, Project Background, Approach to Solving the Problem, Main Results and Analysis, Conclusion, and Opportunities For Future Work.

The Automated Warehouse Scenario was introduced to the Answer Set Programming (ASP) Challenge in 2019. The problem was submitted by Gebser and Obermeier (2019) and involves defining hard constraints in a dynamic world problem. The Automated Warehouse Scenario is based off of the scheduling problem surrounding the delivery of products. The technology of choice used to solve this problem is Clingo, an ASP system which grounds and solves logic programs. For the purpose of this paper AWS will be used in reference to the Automated Warehouse Scenario.

Problem Statement

In AWS, the problem consists of programming the actions of robots in a dynamic environment. The main objective for these robots are to deliver products to picking stations, in turn fulfilling orders. A warehouse is represented as a rectangular grid of X and Y coordinates. The robots can only move vertically or horizontally to adjacent cells per a given time step. An order is considered fulfilled when a given robot carries a shelve with the required products to matching picking station as defined by the order. The overarching goal is to optimize delivery of products by minimizing the time it takes to complete all orders. Time in this context are the actions each robot takes in a given time step. Valid robot actions are move around, pick up and put down shelves, deliver products or remain idle. Additional constraints are that there must be no collisions between robots, a robot carrying a shelf can no longer move under other shelves, and there are designated highway cells that no shelves can be placed on.

Project Background

In terms of Project Background basic understanding came via the 2019 ASP Challenge submission titled "Automated Warehouse Scenario". The use and understanding of ASP and Clingo were necessary tools for solving the AWS problem. External materials that gave hints as to how to model certain actions, their effects, state, and domain independent axioms came from paper Additive Fluents (Lee and Lifschitz, 2001). For example, the use of the cannibal problem to model the state influenced how state modeling should be used for the AWS problem. Value was also found in understanding the aggregates and optimization tools in the language of Clingo. Perhaps the single greatest factor that contributed to a working solution came from the simple framework demonstrated in the Block World example presented by Professor Lee of Arizona State University. The four components consisted of object declaration (initial state), state description (constraints), effects and preconditions of actions (how the state of objects change before and after actions), and domain independent axioms (how the state is intended to persist or their default action if nothing happens). The presented solution uses a modified version of this framework. Major parallels and key concepts are preserved, but the over-aching redesign of the framework is meant to categorize knowledge representation into these four categories: State, Actions, Effects, and Domain Independent Axioms.

Approach To Solving The Problem

Before describing the approach, It's worth noting that the inputs to the problem were provided in the given form:

```
init(object(O,ID),value(at,pair(X,Y))).
% O = any values from the set
% {node, highway, pickingStation, robot, shelf, product, order}
% ID = unique integer for object O
% (X,Y) = coordinate location for object O.
```

The approach to solving the AWS problem was broken into three high level constructs.

- Rules to get the counts of objects
- Rules around locations of objects
- Using Rules to a implement solution

Rules to get the counts of objects

The number of objects was collected through leveraging the #count aggregate in the language of Clingo. Essentially, these rules would count the number of instances passed in and allow quick retrieval to said values. For example, the rule for finding the number of robots is expressed as the following:

```
numRobots(R):-
R=\#count\{I:init(object(robot,I),
value(at,pair(X,Y)))\}.
```

This method was repeated for all objects in the universe. An example use case for a rule of this type was used in defining the limitations that a robot could move in a given warehouse (i.e. leveraging numColumns(X) and numRows(X)).

Rules around locations of objects

Converting the input into manageable rules that surrounded the concept of location and identification for a given object was necessary. For example, if a robot wanted to perform an action, the system must know which robot wants to perform the action, and where that robot is located at time T. Defining the rule which acts as a 'getter function' via an ID for a given object is straight forward.

```
robot(R) :-
init(object(robot,R),value(at,pair(X,Y))).
```

This rule states that the clause robot is true if the R that is provided maps to a robot with the same R passed in as input. A similar rule is created for all objects in the universe. The rule for getting an object at time T is a little more complex and requires that the initial time for all objects be set to 0. Below is an example of this initialization for robot:

```
robotAt(R,object(node,N),0):-
init(object(robot,R),value(at,pair(X,Y))),
nodeAt(N,pair(X,Y)).
```

Essentially this rule initializes the robot at location N at time 0. This initialization is based off those init object functions passed in as input. Objects that would be affected by actions also had function defined for them depending on how they were initialized (at vs on). The key functions defined through similar means were the following:

- pickingStationAt(P,N).
  - Picking station with ID P is at node N.
- robotAt(R,object(node,N),T).
  - Robot with ID R is at node N at time T.
- shelfOn(S,object(node,N),T).
  - Shelf with ID S is place on node N at time T.
- productOn(P,object(shelf,S),with(quantity,Q),T).
  - Product P on shelf S with has a quantity of Q at time T.
- orderAt(O,object(node,N),contains(P,Q),T).
  - Order O at node N has Q amounts of P at time T.

Using Rules to implement a solution

The solution was implemented by breaking down the problem into the following groups and subgroups of human readable sentences mixed with pseudocode :

- State
  - Highway: No picking station on a highway
  - Robot: One robot for one node and robots cannot swap.
  - Shelf: One shelf for one node, one shelf for one robot, and no shelf on two locations (robot vs node)
- Actions
  - robotMove: Robot cannot move outside of the grid
  - pickUpShelf: Shelf cannot be picked up by two robots, a robot cannot pick up two shelves, no two shelves can share a node, and no shelf can be in two locations
  - putDownShelf: A single shelf cannot be put down by two robots, a robot must have a shelf to put it down, and no shelves allowed on a highway.
  - delver: Robot must be on picking station, robot must have the shelf containing product, cannot deliver more quantities than the order, and cannot deliver more quantities than the product.
- Effects
  - Moving a robot: Robot moves to new node at time T+1.
  - Picking up a shelf: Robot stays at same node at time T+1 and shelf gets put on at location of the robot at time T.
  - Putting down a shelf: Robot stays at same node at time T+1 and shelf gets put down at location of the robot at time T.
  - Delivering a product: Order is decremented at time T+1 for the amount of product delivered at time T and the product quantity of the shelf is decremented by the amount of the order.
- Domain Independent Axioms
  - Robot: default to not move at time T+1 if it didn't move at time T.
  - Shelf: defaults to not move at time T+1 if it wasn't picked up/ put down at time T.
  - Order: defaults to having the same quantities if there was no delivery.
  - Product: defaults to having the same quantities if there was no delivery.

With the rules defined for accessing locations and counts of objects, creating the constraints which satisfy the following human readable sentences is straight forward (Examples from each group provided below).

State

"Robots cant swap places."

```
:-  robotAt(R1,object(node,N1),T),
robotAt(R1,object(node,N2),T+1),
robotAt(R2,object(node,N2),T),
robotAt(R2,object(node,N1),T+1),
R1!=R2.
```

Actions

"Shelf cannot be picked up by two robots."

```
:- 2{pickUpShelf(R,S,T): robot(R)}, shelf(S)
```

Effects

"Order is decremented at time T+1 for the amount of product delivered at time T."

```
orderAt(O,object(node,N),contains(P,O-Q),T+1):-
deliver(R,O,with(S,P,Q),T),
orderAt(O,object(node,N),
contains(P,O),T).
```

Domain Independent Axioms

"Robots default to not move at time T+1 if it didn't move at time T."

```
robotAt(R,object(node,N),T+1):-
robotAt(R,object(node,N),T),
not robotMove(R,move(_,_),T),
T<n.
```

Once all human readable sentences are translated into Clingo code, all possible actions given the constraints need to be generated. For example, the following Clingo snippet generates all possible actions for the robotMove action:

```
{robotMove(R,move(X,Y),T):move(X,Y)}1:-
R=1..NUM_ROB, numRobots(NUM_ROB),
T=0..N,N=n-1.
```

This is accomplished through the choice rule and the upper bound limit of 1. Basically it reads, for all robots R, there can be at most one move per robot at a given time. Similar rules are generated for the following actions: pickUpShelf, putDownShelf, and deliver. The final piece is to define the goal state of the system. For the AWS problem, a sufficient terminal state would be all orders are filled (meaning no order needs anymore required product). This final constraint is shown below:

```
:- not orderAt(O,object(node,_),
contains(P,0),n),
orderAt(O,object(node,_),
contains(P,_),0).
```

Optimizations and helpers

In order to find an optimal answer for the given system, the #minimize keyword was used. Specifically this capability was used to minimize the occurrences of actions. In order to track the actions, a helper function was needed. Below is the helper code used to track when the robotMove action would happen:

```
happens(object(robot,R),move(X,Y),T):-
robotMove(R,move(X,Y),T).
```

Similarly all actions would be tracked in this manner, in the form of this triple: happens(Object, Action, Time). This in turn allows for Clingo to optimize on the count of these triples. Thus, minimizing the actions to perform all orders, which in turn optimizes the program:

```
#minimize{1,O,A,T:happens(O,A,T)}.
```

Main Results And Analysis

The following commands were used in order to achieve these results:

- clingo solution.lp inst1.asp -c n=10 -t8
- clingo solution.lp inst2.asp -c n=11 -t8
- clingo solution.lp inst3.asp -c n=7 -t8
- clingo solution.lp inst4.asp -c n=5 -t8
- clingo solution.lp inst5.asp -c n=7 -t8

In order to find the least amount of time required to satisfy the problem, each program began running the command with an n=1. If it was unsatisfiable, n was incremented. This is how n was determined per given ASP instance. Initially the program was ran without the optimization on actions.

Results no Action Optimizations		
Instance	Time(n)	Actions
inst1.asp	10	20
inst2.asp	11	19
inst3.asp	7	12
inst4.asp	5	10
inst5.asp	7	12

The table below is a result of using the optimization on actions for the same ASP instances. The time n was increased up to n+10 to see if more time would enable less actions. This did not prove to be the case.

Results with Action Optimizations		
Instance	Time(n)	Actions
inst1.asp	10	19
inst2.asp	11	17
inst3.asp	7	10
inst4.asp	5	10
inst5.asp	7	10

All instances except inst4.asp had a better solution in terms of the amount of actions performed after the optimization. As seen in the analysis, the increase of time (n) did not result in a more optimal solution in terms of Actions.

Program Return values

inst1.asp

```
happens(object(robot,1),move(-1,0),0)
happens(object(robot,1),move(-1,0),1)
happens(object(robot,2),move(0,-1),1)
happens(object(robot,2),move(1,0),2)
happens(object(robot,1),move(-1,0),3)
happens(object(robot,2),move(0,1),5)
happens(object(robot,1),move(0,-1),6)
happens(object(robot,2),move(0,-1),7)
happens(object(robot,1),move(0,1),8)
happens(object(robot,2),pickup,0)
happens(object(robot,1),pickup,2)
happens(object(robot,2),pickup,6)
happens(object(robot,1),pickup,7)
happens(object(robot,2),putdown,4)
happens(object(robot,1),putdown,5)
happens(object(robot,2),deliver(2,2,1),3)
happens(object(robot,1),deliver(1,1,1),4)
happens(object(robot,1),deliver(1,3,4),9)
happens(object(robot,2),deliver(3,4,1),9)
numActions(19)

OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 64
Calls        : 1
Time         : 0.316s (Solving:0.15s 1st Model:0.14s Unsat:0.01s)
CPU Time     : 1.203s
Threads      : 8         (Winner: 8)
```

inst2.asp

```
happens(object(robot,1),move(-1,0),0)
happens(object(robot,1),move(-1,0),1)
happens(object(robot,2),move(1,0),1)
happens(object(robot,2),move(0,-1),2)
happens(object(robot,1),move(-1,0),3)
happens(object(robot,1),move(1,0),5)
happens(object(robot,2),move(0,1),5)
happens(object(robot,2),move(-1,0),7)
happens(object(robot,2),move(-1,0),8)
happens(object(robot,2),move(0,1),9)
happens(object(robot,2),pickup,0)
happens(object(robot,1),pickup,2)
happens(object(robot,2),pickup,6)
happens(object(robot,2),putdown,4)
happens(object(robot,2),deliver(2,2,1),3)
happens(object(robot,1),deliver(1,1,1),4)
happens(object(robot,2),deliver(1,3,2),10)
numActions(17)

Models       : 2
  Optimum    : yes
Optimization : 72
Calls        : 1
Time         : 0.361s (Solving:0.23s 1st Model:0.05s Unsat:0.14s)
CPU Time     : 1.588s
Threads      : 8        (Winner: 8)
```

inst3.asp

```
happens(object(robot,1),move(0,-1),0)
happens(object(robot,1),move(-1,0),1)
happens(object(robot,1),move(0,-1),3)
happens(object(robot,2),move(0,-1),3)
happens(object(robot,1),move(1,0),5)
happens(object(robot,2),move(1,0),5)
happens(object(robot,1),pickup,2)
happens(object(robot,2),pickup,2)
happens(object(robot,1),deliver(2,4,1),4)
happens(object(robot,2),deliver(1,2,1),6)
numActions(10)

Models       : 1
  Optimum    : yes
Optimization : 31
Calls        : 1
Time         : 0.065s (Solving:0.01s 1st Model:0.01s Unsat:0.00s)
CPU Time     : 0.145s
Threads      : 8        (Winner: 8)
```

inst4.asp

```
happens(object(robot,1),move(-1,0),0)
happens(object(robot,1),move(-1,0),1)
happens(object(robot,2),move(0,-1),1)
happens(object(robot,2),move(1,0),2)
happens(object(robot,1),move(-1,0),3)
happens(object(robot,2),pickup,0)
happens(object(robot,1),pickup,2)
happens(object(robot,2),deliver(3,2,2),3)
happens(object(robot,1),deliver(1,1,1),4)
happens(object(robot,2),deliver(2,2,1),4)
numActions(10)

Models       : 1
  Optimum    : yes
Optimization : 20
Calls        : 1
Time         : 0.058s (Solving:0.01s 1st Model:0.01s Unsat:0.00s)
CPU Time     : 0.088s
Threads      : 8        (Winner: 8)
```

inst5.asp

```
happens(object(robot,1),move(-1,0),0)
happens(object(robot,1),move(-1,0),1)
happens(object(robot,1),move(-1,0),3)
happens(object(robot,2),move(-1,0),3)
happens(object(robot,1),move(0,1),5)
happens(object(robot,2),move(0,1),5)
happens(object(robot,1),pickup,2)
happens(object(robot,2),pickup,4)
happens(object(robot,1),deliver(1,1,1),4)
happens(object(robot,2),deliver(1,3,4),6)
numActions(10)

Models       : 2
  Optimum    : yes
Optimization : 31
Calls        : 1
Time         : 0.091s (Solving:0.02s 1st Model:0.01s Unsat:0.00s)
CPU Time     : 0.196s
Threads      : 8        (Winner: 8)
```

Conclusion

In conclusion, there was a satisfiable solution for all five instances provided for the AWS problem. The general framework for solving the problem came through understanding how to model and represent knowledge. Specifically, the problem had to be defined in terms of state constraints, actions, effects of said actions, and domain independent axioms. By defining in Clingo what the problem was through rules and ASP concepts, the computer was able to generate the proper solutions according to the constraints. In contrast to declarative programming that knows nothing about the problem, but actively tries to solve it through traditional algorithm design and programming methodologies, knowledge representation requires the definition of constraints that model human readable sentences of the problem. All possible actions are created, then the search space is pruned through constraints defined by the knowledge representations. This was exemplified through AWS by translating the problem (a collection of human readable sentences) into a language the computer could understand (Clingo/ASP) then allowing the computer to arrive at potential solutions based off of the understanding of the problem, thus demonstrating the goal of Knowledge Representation.

Opportunities For Future Work

There are definitely more potential experiments/optimizations for the AWS problem. Here are a few examples of exploration that may be considered.

Optimal number of Robots

Assume a certain warehouse is trying to keep operating costs to a minimum. Instead of providing the amount of robots initially (like the given instances), the program would be required to find the optimal amount of robots based off of a certain cost threshold.

Optimal initial positions of shelves

Perhaps a warehouse has a worker assigned to rearranging the shelves based on the orders required the following day. The problem could then be modified to find the optimal position of shelves given the future orders and the highway constraints.

Serializable solution

Perhaps the robots are not fully functional to react at the exact moment necessary, so a certain amount of slack is required between their actions. A fun optimization to the problem would be to find a serializable solution, meaning all actions can happen independent of each other.

Clingo vs Prolog

Provide a solution for both Clingo and Prolog languages and compare run times and optimal solutions that each language provides. Perhaps some underlying technology may find a more optimal solution. Perhaps a combination of both would provide and even greater and undiscovered methodology for the AWS problem.

References

Gebser, M. and Obermeier P. 2019. ASP Challenge Problem: Automated Warehouse Scenario. Klagenfurt Univ., Graz Univ., Postdam Univ.

Lee, J. and Lifschitz V. 2001. Additive Fluents, Thesis, Department of Computer Sciences, University of Texas, Austin, TX.