Solving the Automated Warehouse Scenario using Answer Set Programming

Eric Waters
School of Computing and Augmented Intelligence
Arizona State University
eswaters@asu.edu

Problem Statement

The Automated Warehouse Scenario (Gebser and Obermeier 2019) provides a great opportunity to demonstrate the capabilities of Answer Set Programming (ASP). This particular problem was created as part of the ASP Challenge 2019, which was coordinated by the University of Genoa and the Technische Universitat Wien. The aim of the challenge is to bring in members of the ASP community and businesses to solve real-world problems faced by various industries.

The scenario is representative of an automated retail warehouse. The warehouse is defined as a rectangular grid of tiles. On these tiles exists one or more robots that have the capability of moving in any compass direction. Spread across the warehouse are shelves that contain a given quantity of products. There is a number of orders that demand certain quantities of certain products. Robots can fit under the shelves (if they are not already carrying one) in order to pick them up. The goal is to have the robots deliver the products to the appropriate stations in their designated quantities, therefore fulfilling the order. The order fulfillment should be optimized such that it is completed in the minimum amount of time possible. In this scenario, each robot is restricted to one action (moving, picking up a shelf, putting down a shelf, or delivering a product to a picking station) per unit of time. Shelves must not be put down on tiles designated as a highway. Lastly, robots must be prevented from colliding with each other.

Optimization scenarios such as the Automated Warehouse are common in a wide range of industries. ASP is a useful tool in solving such optimization problems as it allows for the efficient modeling of complex constraints and preferences. The ASP Challenge 2019 provides an excellent platform for members of the ASP community to showcase their skills and collaborate with businesses to solve real-world problems. With the rise of automation and the need for efficient logistics in various industries, scenarios like the Automated Warehouse are becoming increasingly important.

Project Background

Knowledge Representation and Reasoning (KRR) is a subset of Artificial Intelligence that focuses on representing knowledge in a way that computers can comprehend and utilize to make decisions. Models of knowledge can be expressed using various techniques such as Propositional Logic or First-Order Logic. KRR techniques can be used in various applications including robotics and natural language processing.

A major issue with first-order logic is that it is undecidable, i.e., no algorithm can always terminate and determine whether or not an arbitrary first-order formula is satisfiable. This is due to the fact that first-order logic can quantify variables over infinite domains. Answer Set Programming addresses this issue by providing a subset of first-order logic that is tractable.

ASP allows the specifications of rules and constraints that describe the problem. For example, in order to describe the fact that the robot in the Automated Warehouse Scenario must be located somewhere in the warehouse, the constraint could be formalized as in Listing 1.

Listing 1: Robots at Valid Tiles

```
:- robot(R,X,Y,T), not node(_,X,Y).
```

This rule states that if a robot R is at position X and Y at time T, and those X and Y values do not belong to a node (warehouse tile), then the program is violated. ASP will not return solutions that break this rule.

After the program is defined, the ASP solver will find all solutions such that no constraints are violated. These solutions are called stable models of the program.

For this Automated Warehouse Scenario, I use the ASP language of Clingo and their ASP solver. Clingo is one of many ASP languages/solvers and is known for its high-performance solving and support for advanced features such as aggregates and optimization, which proves very useful for the Automated Warehouse Scenario. There may be thousands of solutions to the scenario, but the desired solution is the one that has the fewest amount of actions in the smallest number of units of time.

Approach to Solving

My approach to solving the Automated Warehouse Scenario is a combination of several techniques.

The first technique is the Generate-Define-Test method as explained by Dr. Joohyung Lee in his lectures on Answer Set Programming (Lee 2023a). This separates the process of writing an ASP program into three parts.

1. Generate: In this stage, I generate all scenarios. To put this in terms of the Automated Warehouse Scenario, this would mean that robots perform all possible actions. They move everywhere, pick up every shelf, etc. This is also called generating a ”search space”.
2. Define: In this stage, I define new fluents. Fluents are atoms that depend on the state of the program. For example, I create a fluent called robot(R,X,Y,T) which says that a robot R is located at position (X,Y) at time T. I do the same for shelves, products, orders, highways, etc.
3. Test: In this stage, I write constraints to invalidate elements of the search space that I do not want to be possible. For example, refer back to Listing 1 where I write the constraint that a robot must be within the pre-defined grid of warehouse tiles.

The next technique comes from Dr. Joohyung Lee in his lectures on Reasoning about Actions (Lee 2023b). In these lectures, he explains how to describe actions in Answer Set Programming. For example, how would one formalize the action of the robot moving? Lee explains the idea of transition systems, which are directed graphs in which the vertices correspond with states and the edges correspond with actions. Essentially what I have created for the Automated Warehouse Scenario is a very complex transition system. There is an initial provided state. If actions occur, such as a robot moving, that action transitions the program to a new state. The goal state is the one in which all of the products are delivered to their appropriate picking stations with the least amount of actions and time elapsed.

Defining this transition system for the Automated Warehouse Scenario was a complex process and will be better understood if broken down into several sections.

The Initial State

The initial states of the program were provided by ASP Challenge 2019. These were five test scenarios of different initial configurations of the warehouse.

- The warehouse was defined as a 4x4 grid.
- Some of these tiles were marked as highways, which are locations in which the robot must not put down a shelf in order to allow other robots to move through it more easily.
- Some tiles were designated as picking stations. These tiles are the ones where the robot must deliver the shelves containing the products.
- A number of robots and their initial locations were defined.
- A number of shelves and their initial locations were defined.

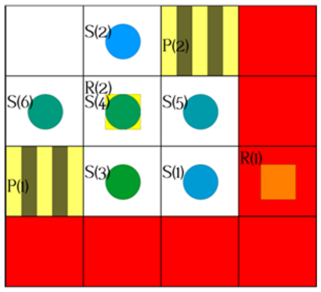


Figure 1: One of the initial configurations for the warehouse. Squares labeled with R represent robots. Circles labeled with S represent shelves. Yellow-striped tiles represent picking stations. Red tiles represent highways.

- A number of products were defined. This included which shelf the product was located on and how many units of that product existed on that shelf.
- A few orders were defined. The order included:
  - Which products were to be included in the order
  - The quantity of each product to be included in the order
  - The picking station where the ordered must be fulfilled

One of the initial configurations is shown in Figure 1. I took this initial configuration and turned them into my own fluents. For example, for the initialization of the location of the robots, I created the rule shown in Listing 2.

Listing 2: Robot Initialization

```
init(object(node,1),value(at,pair(1,1))).
robot(R,X,Y,0) :- init(object(robot,R),value
    (at,pair(X,Y))).
```

The first line, provided by ASP Challenge 2019, is their way of providing the initialization. To make this easier for my program to handle, I turned it into a robot(R,X,Y,T) fluent where R is the ID of the robot and X and Y are the coordinates of its location at time T. I followed this same principle for all of the initialized objects.

Domain-Independent Axioms

Regardless of the problem at hand, there are certain axioms that should be defined in order to create a robust transition system.

Fluents are Unique and Guaranteed to Exist At each time step T, each fluent should exist and be unique. In terms of the Automated Warehouse Problem, an example is the idea that each robot should be in exactly one location at each time T. The rule to represent this idea is shown in Listing 3.

Listing 3: Robots Guaranteed to Exist and Unique

```
:- not 1 {robot(R,X,Y,T)} 1, robot(R), T=0..
    m.
```

Actions May or May Not Happen At each time step T, each action might occur depending on some specified criteria. This allows Clingo to consider all possibilities of valid actions in order to search for all potential solutions to the problem. For example, consider the following rule in Listing 4:

Listing 4: Robots May or May Not Pick Up a Shelf

```
{occurs(object(robot,R),pickup,T)} :- robot(
    R,X,Y,T-1), shelf(S,X,Y,T-1), T=1..m.
```

This rule states that if a robot R is at position (X,Y) at time T-1 and there is also a shelf at that same (X,Y) location at time T-1, the robot may or may not pick up the shelf at time T. Similar rules were created for all other possible actions.

The Commonsense Law of Inertia This law of inertia simply defines the states of the program if an action is not executed. For example, if a robot does not perform a move action, it should stay in its same location at the next time step T. This rule is shown in Listing 5.

Listing 5: Robots Stay In Place Unless Moved

```
{robot(R,X,Y,T+1)} :- robot(R,X,Y,T), T<m.
```

Similar rules were defined for all other fluents in order to specify their states at the next time step if no action occurs.

Prerequisite Conditions and Effects of Actions

In this section, I define the criteria that must be met in order for a specific action to occur. For example, shown in Listing 6 are the effects and rules for a robot moving:

Listing 6: Definition of Robot Movement

```
robot(R,X+Dx,Y+Dy,T) :- robot(R,X,Y,T-1),
    occurs(object(robot,R),move(Dx,Dy),T).
```

This rule states that a robot R will update its position to (X+Dx,Y+Dy) at time T if the robot R exists at (X,Y) at time T-1 and the move(Dx,Dy) action occurs at time T. Similar rules were defined for the other actions (picking up a shelf, putting down a shelf, and delivering products).

State Constraints

This section of the program defines states that should be impossible. This is the ”Test” stage outlined in the aforementioned Generate-Define-Test methodology. It is crucial to invalidate stable models that do not meet the criteria of the problem at hand. For example, Listing 7 is a constraint that prevents two different robots from being in the same location at the same time (a collision):

Listing 7: Two Robots Cannot Collide

```
:- robot(R1,X,Y,T), robot(R2,X,Y,T), R1!=R2.
```

Below is the list of the constraints that I used in order to define impossibilities in my program:

- Robots cannot be located at invalid positions in the warehouse
- Two robots cannot be in the same position at the same time (collision)
- Robots cannot cross (switch places) - this would also represent a collision
- If a robot carries a shelf, no other shelf can be at the same location at the same time (shelf collision)
- A robot cannot be in two different positions at the same time
- A shelf cannot be in a different position as a robot if that robot is carrying that shelf
- A robot cannot perform more than one action per time step
- A product cannot have two different quantities at the same time step
- An order item cannot have two different remaining quantities needed at the same time step

Goal Constraint and Optimization

In this section, the goal of the problem was defined. The goal was to have all of the orders fulfilled by a given time T, i.e., the remaining quantities for each order item must be zero. In order to optimize the problem, the number of actions A must be kept to a minimum. This was achieved using the rules in Listing 8.

Listing 8: Fulfill Orders and Minimize Actions

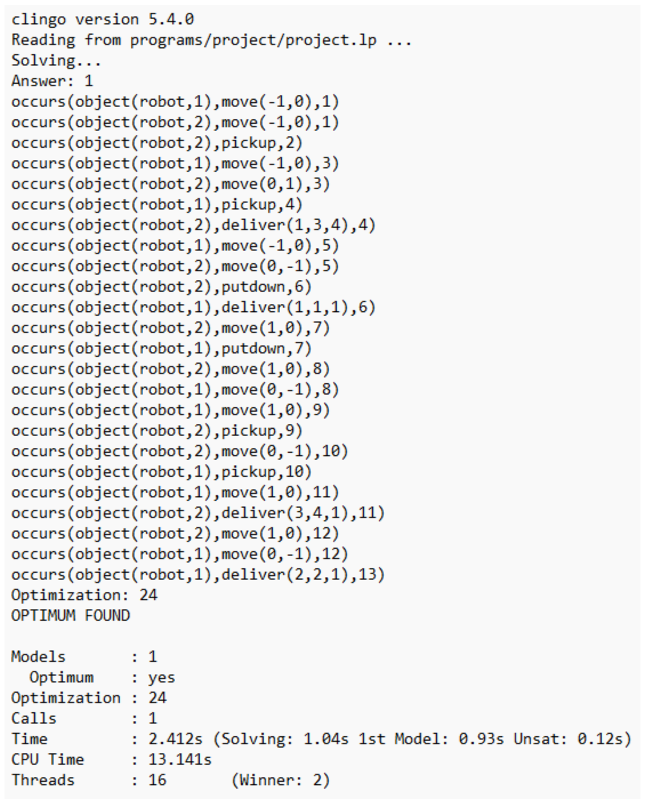
```
:- orderItem(O,I,U,T), U>0, T=m.
#minimize {1,R,A,T : occurs(object(robot,R),
    A,T)}.
```

Results and Analysis

The program functions as expected and provides the optimal set of actions to deliver all of the necessary products in their appropriate quantities to the appropriate picking stations in the least amount of time. The output of the program for the first test scenario provided by ASP Challenge 2019 is shown in Figure 2. Notice how the program took just over two seconds to complete on a 16-thread processor. This is a reasonable amount of time for the optimal action plan to be found in a real-world warehouse, especially considering the time wasted if a non-optimal plan were to be followed. This demonstrates that Answer Set Programming is a viable and effective option for discovering optimal solutions to complex logistical problems. Furthermore, the program is very manageable to write, with only 90 lines of code including comments or 53 lines of code without comments. Clingo has enabled the solving of this Automated Warehouse Problem to be efficient and concise.

Conclusion

In this work, I demonstrate the capabilities of Knowledge Representation and Reasoning (KRR) and Answer Set Programming (ASP) to solve a complex logistical problem, the Automated Warehouse Scenario provided by the ASP Challenge 2019. ASP is an ideal tool for addressing such optimization problems because they allow an efficient means of modeling the constraints and rules.



```
clingo version 5.4.0
Reading from programs/project/project.lp ...
Solving...
Answer: 1
occurs(object(robot,1),move(-1,0),1)
occurs(object(robot,2),move(-1,0),1)
occurs(object(robot,2),pickup,2)
occurs(object(robot,1),move(-1,0),3)
occurs(object(robot,2),move(0,1),3)
occurs(object(robot,1),pickup,4)
occurs(object(robot,2),deliver(1,3,4),4)
occurs(object(robot,1),move(-1,0),5)
occurs(object(robot,2),move(0,-1),5)
occurs(object(robot,2),putdown,6)
occurs(object(robot,1),deliver(1,1,1),6)
occurs(object(robot,2),move(1,0),7)
occurs(object(robot,1),putdown,7)
occurs(object(robot,2),move(1,0),8)
occurs(object(robot,1),move(0,-1),8)
occurs(object(robot,1),move(1,0),9)
occurs(object(robot,2),pickup,9)
occurs(object(robot,2),move(0,-1),10)
occurs(object(robot,1),pickup,10)
occurs(object(robot,1),move(1,0),11)
occurs(object(robot,2),deliver(3,4,1),11)
occurs(object(robot,2),move(1,0),12)
occurs(object(robot,1),move(0,-1),12)
occurs(object(robot,1),deliver(2,2,1),13)
Optimization: 24
OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 24
Calls        : 1
Time         : 2.412s (Solving: 1.04s 1st Model: 0.93s Unsat: 0.12s)
CPU Time     : 13.141s
Threads      : 16       (Winner: 2)
```

Figure 2: The output of the Clingo program, an optimized action plan for the first provided test scenario. It can be done in 24 actions and 13 time steps.

The approach to solving this problem using ASP is a combination of several techniques, including the Generate-Define-Test method and the idea of describing actions. Careful attention to detail is required when performing Answer Set Programming. All impossible states must be considered and properly defined. Furthermore, all actions must be explicitly defined, including their preconditions and effects.

The program functions as expected and provides the optimal set of actions to deliver all of the necessary products in their appropriate quantities to the appropriate picking stations in the least amount of time. The output of the program for the first test scenario provided by ASP Challenge 2019 shows that the program took just over two seconds to complete on a 16-thread processor. This is a reasonable amount of time for the optimal action plan to be found in a real-world warehouse, especially considering the time wasted if a non-optimal plan were to be followed.

Opportunities for Future Work

Although the program provides the optimal solution to the Automated Warehouse Scenario, there are areas where future work could further enhance the program's effectiveness in a real-world scenario.

First, the orientation of a real warehouse is not necessarily static. If the warehouse or the inventory is to change, there should be little to no modifications required to the code. However, in this program, the initial orientation of the warehouse, the products, and the orders must be carefully and explicitly defined. Either programs would need to be written to automatically generate the new warehouse and product orientations or the program itself would have to be able to handle these dynamics itself.

Additionally, more testing would have to be done in terms of the scalability of this program. The provided test scenarios are only 4x4 grids, which is unrealistic; real-world warehouses are much larger and have many more products and complexities. Testing on real-world scenarios and further optimization would be necessary to implement this for an organization.

In general, the Automated Warehouse Scenario is an over-simplification of the complexities and scale of a real-world warehouse. Those complexities would need to be accurately represented in the system in order for it to be implemented in practice. However, this program serves as a proof of concept that Answer Set Programming is indeed a viable option for optimizing complex logistical problems and should be considered as an option by industries attempting to implement solutions to similar problems.

References

Gebser, M., and Obermeier, P. 2019. Asp challenge problem: Automated warehouse scenario.

Lee, J. 2023a. Practice of answer set programming. Lecture video series for Knowledge Representation and Reasoning course.

Lee, J. 2023b. Reasoning about actions. Lecture video series for Knowledge Representation and Reasoning course.