Project Report

for Amazon Automated Warehouse

Anubhab Guha
Arizona State University
aguha4@asu.edu

Daniel Pilgrim-Minaya
Arizona State University
dpilgrim@asu.edu

Kanisk Bashyam
Arizona State University
kbashyam@asu.edu

Yi-Chuan Lin
Arizona State University
ylin269@asu.edu

Abstract

For the course CSE 579 Knowledge Representation taught by Zhun Yang at Arizona State University, our team demonstrated our skills by doing a semester-long course project. For this course project, our team solved the automated warehouse problem, which is a dynamic world problem previously posted by Amazon for the ASP competition. In this project, our team utilized our clingo programming abilities to find the optimal plan for each professor-given warehouse scenario to fulfill all orders in a minimum amount of time.

Problem Statement

The scenario of the project consists of robots which are assigned orders to deliver products to their designated picking station. The map of warehouse is defined by a rectangular grid of nodes or cells where the robot can move vertically and horizontally to adjacent cells. To fulfill the orders assigned to the robots, the robots need to pick up shelves with the designated products and carry the shelves to their respective picking stations. The robots are flat and can move underneath shelves. But when a robot is carrying a shelf, it cannot fit under another shelf and can only travel through the path which is defined as the highway, where a robot cannot put down shelves. So, if the way of the robot is blocked, it first needs to move the shelves out of its way. The goal of the project is to fulfill the orders assigned to the robots in the least possible time, where time is counted in steps and a robot can perform at most one action at each step. There is also a constraint that two robots cannot switch their position with one another in a single time step as it is considered as a collision.

Project Background

Our project is implemented using clingo which employs Answer Set Programing in order to solve NP-hard class of search problems. Answer Set Programming solves this by reducing the problem to stable model which then uses answer set solver to perform the desired search functionality. These answer set solvers are essentially programs to calculate a set of stable models. These programs are created by enhancing the DPPL algorithm and terminate unlike programs in Prolog.

Additionally, the resource we used primarily followed the video lectures available on Canvas. The warehouse scenario implementation was very similar to that of the block world problem. Hence, we used that scenario as a way to structure our actions. Similarly, concept such as exogeneity and inertia were defined with the help of our course material and the clingo manual.

Finally, the last resource we utilized revolved around testing of our stable models. Since the actual scenarios generated were very difficult to visualize, we utilize a visualizer called asprilo. This system was specifically designed in order to view designs inspired by an automated warehouse system that was built using multiple moving robots and shelves. This enabled us to view the actions performed by the robot during each timestamp and verify our findings.

Our Approach Towards Solving the Problem

Our first meeting was organized by our team to go through the problem statement and assign each member of our group with a specific task to perform. Following that a Github repository was setup where each member created their own branches to eliminate the problem of erroneous code.

The grid world of the problem was instantiated before starting to implement the specific actions assigned to the respective team members. Instances of the highway, robot, shelf, picking station, and product was also defined.

After the instances were setup, each member started implementing the actions assigned to them and committing to their respective branches on Github. Different test cases were designed for each of the actions defining a stripped-down version of the grid world and assigning simple goals to fulfill.

Subsequent to the implementation of the actions delegated to each team member, we started putting the actions together in one file under warehouse.lp to try to make the actions work collectively. After numerous testing and thorough debugging, we were able to obtain solutions for all six of the instances provided for the project.

Results and Analysis

The asprilo tool enabled us to visualize the scenarios generated by clingo. We implemented the scenario by specifying the various actions and instances strictly according to the project description provided to us. The stable models generated provided the list of all the actions the robot performed leading to the all the orders being fulfilled. This was calculated by storing the number of items specific to a product within an atom named order. This atom would change every time the action deliver occurred. Thus, this atom was our main factor for determining our terminating condition. The results obtained from running the clingo program could be transformed into the format specific to the asprilo system, thus enabling us to visualize the set of actions that take

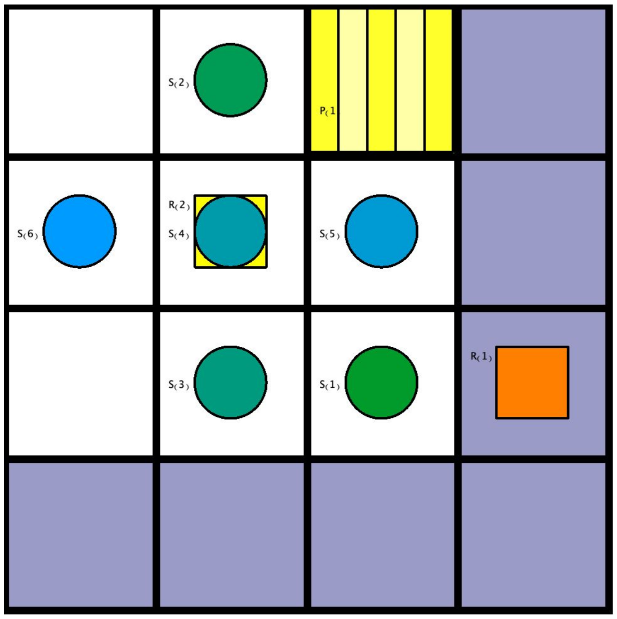
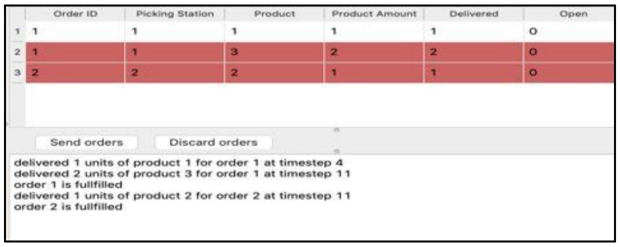


Figure 1. A visualized instance setup (inst3 from simple instances).



	Order ID	Picking Station	Product	Product Amount	Delivered	Open
1	1	1	1	1	1	0
2	1	1	3	2	2	0
3	2	2	2	1	1	0

```
delivered 1 units of product 1 for order 1 at timestep 4
delivered 2 units of product 3 for order 1 at timestep 11
order 1 is fullfilled
delivered 1 units of product 2 for order 2 at timestep 11
order 2 is fullfilled
```

Figure 2. Steps of order being fulfilled visualized with output of inst3.

Warehouse Scenario	Minimal Timesteps Solved
1	13
2	11
3	7
4	10
5	6

Table 1. Minimal timesteps for solving all simple instances.

place. Once this was in place, we were then able to verify all the constraints that were meant to be placed on the robot. For example, a robot never putdown a shelf on the

highway, never went under a shelf while carrying another shelf.
Initially, we faced issues while implementing the "deliver" functionality. The constraints imposed were not enough making the program inefficient and would end up within an infinite loop. This needed to be reconstructed, we added further conditions which enabled the robot to perform only one action at a particular time stamp. Additionally, two types of deliver's needed to be instantiated, one where the items delivered were less than equal to the required order and the second where the number of items on the shelf were more than the current order required. Further by adding constraints limiting the number of items being delivered to be more than zero but less than equal to the actual requirement lead us to a terminating state at an optimal time.

Issues Encountered

The first issue we faced while doing the project was the integration of the different parts of the project implemented by different team members. To solve that problem, a Github repository was setup. But due to testing method of the different actions in clingo, the members had to update the instances to test their specific actions. To solve the problem of tackling with multiple instances, the members created their own branches of the repository where each member had their own instances geared towards testing their own clingo programs.

While implementing the problem we faced a doubt with what happens to a node when a robot picks up a shelf from that node. Our team got confused whether to classify that now empty node as a highway or just a node in general. But after emailing our instructor, it became clear that it just becomes an "empty node" when the shelf is picked up.

Later on, in the testing phase of our project we encountered a problem with the "move" action as the robot was getting out of the defined nodes of the grid world. So, a constraint was put in place to solve the problem.

Individual Contribution

Anubhab Guha: As the action assigned to Anubhab was "move", he built the prototype of the action and pushed it to the repository. He made iterative updates for his action so that it keeps coherent with the rest of the project. Additionally, he wrote test cases to test his action separately.
After the project was implemented, the final phase of the project, i.e., documentation, was delegated among the team members. Anubhab had a major contribution towards writing the final report for the project.
He attended all the group meetings, including two online meetings on Google Hangouts, and one in person meeting.

Daniel Pilgrim-Minaya: Daniel created the Github repository so that the team members are able to collaborate for the group project.
Daniel's task was to implement the "pickup" action. He built the prototype for his task, pushed it to the repository, and made frequent updates to it. He also created a testing module for all the actions, so that all the actions can be tested for consistency.
He led discussions and scheduled team meetings to keep the team members updated with the progress that has been made.

Kanish Bashyam: Kanishk was assigned to implement the deliver action. The deliver action was first design by enumerating all the necessary condition around it. Additionally, Kanishk helped visualize the scenarios using the asprilo system. Using the visualizer helped testing the result Lin had obtained. Further, contributed to the report and helped other teammates wherever required.

Yi-Chuan Lin: Lin was assigned to implement the "putdown" action for the robot. She built the prototype for "putdown" competently and kept the Github repository updated so that other team members can test the coherence of their own code.
Apart from materializing her own given tasks, she instantiated the grid world of the problem so that the group could have a common ground to discuss and develop the program. Implemented fluents' inertia, value constraints and goal for the program. Furthermore, she played a crucial role in the debugging phase of the project and helped in making our solutions satisfiable.
She also took an active part in all the group meetings.

Opportunities for Future Work

This project scenario is a simplified version of the Amazon warehouse. So, for future work, this problem can be developed further to represent the Amazon warehouse with more structural accuracy.

We can expand the grid world to account for moving warehouse employees. An additional "restock" action can be implemented which will restock the shelves with the necessary products when a shipment truck arrives at the warehouse. Consequently, we can add a "restock station" where the robots will carry the shelves which need to be restocked.

Conclusion

For this course project, we were given a simplified version of multiple Amazon Warehouse Scenarios to create a Program to solve these scenarios by fulfilling all orders in as minimal time possible. Given our knowledge of Answer Set Programming and Clingo, we were able to construct a solver that we could apply to these scenarios to develop a solution. Using Test-driven development, peer review, and proper labeling of clingo code, our team members were able to create this solver program. The solver program contains definitions and constraints for object initializations (nodes, highways, pickingstations, robots, shelves, and products), robot actions (move, pickup, putdown, and deliver), value constraints, and inertia. Our program's goal was to output a solution for all orders being fulfilled. Clingo was able to adequately solve each warehouse scenario and produce a minimal timestep plan to fulfil all orders. We achieved the goal and calculated the minimum number of timesteps for each scenario. For future work, we aim to move past the simplified version of the amazon warehouse and define objects for human workers, warehouse resupply orders, and unloading trucks. We are grateful to our professor, Zhun Yang, for helping us achieve this goal.