Automated Warehouse Scenario

Adarsh Saripalli[*]
Arizona State University
asaripa3@asu.edu

Abstract

Robots, such as Kiva robots, now discover shelves and carry them to packing stations more quickly and effectively than humans did in the past when they had to go around a large warehouses. Robots like this must be able to move fast, avoid collisions, adhere to regulations, and handle shelves with caution. If you try to figure out every move they could make in advance, it can be slow and very difficult to plan their motions. Answer Set Programming (ASP) is a solution for it. By providing robots with a set of information and guidelines about the warehouse, ASP allows a smart system to determine the most efficient path for them to take. Even when the warehouse is big or changing often, it helps the robots identify good, quick solutions. Today's robots can safely, reliably, and swiftly fulfill orders because of ASP and other intelligent planning techniques.

The goal of this project is to mimic automated warehouses, where robots traverse a grid-based system to retrieve shelves and transport goods to designated picking stations. Using ASP to fulfill client requests as fast as feasible while controlling a limited supply of products, avoiding crashes, and adhering to traffic limits is the aim.

Problem Statement

As businesses expand larger, they can no longer rely only on employees to complete all of the jobs by hand since people are fallible, prone to errors, and unable to work continuously. Robots such as Amazon's Kiva are unique in that they can lift huge shelves, never get fatigued, and do the same jobs precisely throughout the day. But getting robots to work on their own in a big warehouse that's always changing is not easy. In this case, the challenge is setting up a system inside a warehouse that looks like a giant grid. The robots have to figure out where the shelves with the products are and carry them to the right packing stations as quickly as possible. They must carefully arrange their movements to avoid colliding with one another because there could be multiple robots functioning together simultaneously. They must transport, pick up shelves, and even take breaks without creating any issues. Additionally, robots must adhere to certain ”traffic rules” in warehouses, such as avoiding parking in specific areas (like highways) and avoiding colliding with another robot that is carrying a shelf. Robots must promptly and safely alter their routes since the warehouse layout is subject to change, including the possibility of shelves moving or robots ending up in unexpected locations. The primary objective is to develop a comprehensive system that directs the robots on precisely which actions to execute in what order so that they can satisfy every customer requirement as quickly, safely, and effectively as feasible. If this issue is resolved, warehouses will be able to handle orders far more quickly, make greater use of their space, and become safer workplaces. Businesses would also find it simpler to expand and conduct their business in a more sustainable manner. In order to maximize the potential of these intelligent robots in the actual world, it is crucial to develop sophisticated systems that are capable of autonomous reasoning via knowledge-based representations.

Project Background

In order to address logistical challenges, manual work in warehouses was replaced by robots. However, it proved to be quite challenging to get robots to operate autonomously in expansive, dynamic situations. Robots were initially instructed to move forward, turn, or pick up objects, but as more were added, it became evident that more complex techniques were required to keep the robots from colliding and becoming trapped. Early solutions like **Backtracking** and **Pathfinding** were used to explore different routes and correct mistakes, but they quickly became too slow and complicated as the warehouse systems grew. Robots must constantly comply to rules like avoiding collisions and moving shelves efficiently, which gave rise to the concept of **Constraint Satisfaction Problems** (CSPs). Even still, conventional methods found it difficult to handle the increasing complexity, which paved the way for declarative programming via ASP.

With ASP, engineers only needed to define the rules, de-

[*]This ASP project was submitted by Adarsh Saripalli for the course CSE 579: Knowledge Representation and Reasoning during the Spring 2025 term, in lieu of coursework.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

scribe the warehouse, and describe the algorithm the objective; the ASP solver would then determine the optimal plan of actions without any manual intervention. Key concepts in ASP included:

- **Choice Rule**, which allows choosing exactly one action (moveUp, moveDown, etc.) at each time step
- **Cardinality Constraints**, which limits how many facts (e.g., pickups) can be true simultaneously
- **Common Sense Law of Inertia**, which states that things remain the same unless something is done about them.
- **Optimization**, which finds the best solutions

Building on these notions, I used the **ASP Challenge 2019** as my major source of inspiration for designing my own warehouse domain by taking account of all the logical constraints needed for autonomous planning. Also, I organized the ASP into distinct domain and problem files like the PDDL approach. The prior Blocksworld challenge assignment worked as the skeleton in framing my approach.

Approach towards Solution

When I first began working on the project, my natural instinct was to sketch the layout of the warehouse instances and gradually formulate the necessary constraints as my understanding of the system deepened. Breaking down the `inst1.asp` file and interpreting the warehouse setup provided a clear visual representation, as shown in Figure 1

It became evident that separating the automated warehouse system into a **domain file** and a **problem file** would lead to a cleaner and more modular structure. The domain file encapsulates all the logical constraints required to find an optimal plan for the robots to deliver products and reach the goal state. Meanwhile, the problem file represents each specific warehouse scenario with different arrangements and requirements.

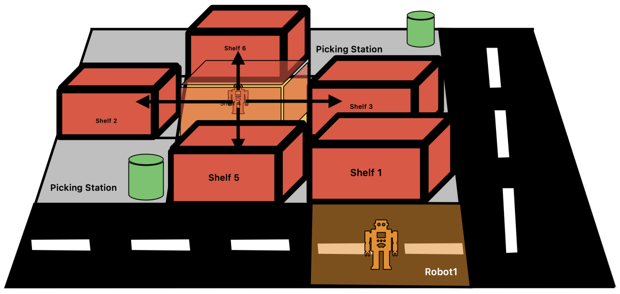


Figure 1: A 4x4 grid layout of an Automated Warehouse.

Problem File Design

The warehouse instance is represented as a 4x4 grid comprising 16 nodes, each node corresponding to a specific position. The key components of the problem setup are as follows:

- 2 robots with IDs 1 and 2.
- 6 shelves with IDs 1 to 6.
- 4 products with IDs 1 to 4.
- 2 picking stations with IDs 1 and 2.
- 3 customer orders with IDs 1 to 3.
- 4 rows and 4 columns

Node 1	Node 5	Node 9	Node 13
Node 2	Node 6	Node 10	Node 14
Node 3	Node 7	Node 11	Node 15
Node 4	Node 8	Node 12	Node 16

The layout is as follows:

- Highways are located at nodes 4, 8, 12, 13, 14, 15, and 16.
- Picking Station 1 is located at node 9, and Picking Station 2 at node 3.
- Robot 1 starts at node 12, and Robot 2 at node 6.
- Shelf placements:
  - Shelf 1 at node 11.
  - Shelf 2 at node 2.
  - Shelf 3 at node 10.
  - Shelf 4 at node 6.
  - Shelf 5 at node 7.
  - Shelf 6 at node 5.
- Product allocations:
  - Product 1 is on Shelf 3 (quantity: 1).
  - Product 2 is on Shelf 4 (quantity: 1).
  - Product 3 is on Shelf 6 (quantity: 4).
  - Product 4 is on Shelf 5 (quantity: 1) and also on Shelf 6 (quantity: 1).

The final goal is to deliver all required products to the respective picking stations:

- Order 1: Deliver Product 1 (qty 1) and Product 3 (qty 4) to Picking Station 1.
- Order 2: Deliver Product 2 (qty 1) to Picking Station 2.
- Order 3: Deliver Product 4 (qty 1) to Picking Station 2.

To encode this information, the following functions were defined in the problem file:

- `nodeAt()`, `pickingStationAt()`, `robotAt()`, `shelfOn()`,`productOn()`,`orderAt()`.

These functions specify the number of robots, shelves, rows, columns, nodes, and orders, assign ID values, allocate nodes to grid positions, place shelves, define highways and picking stations, map products onto shelves with their quantities, and finally associate orders with picking stations and required products. This marks the completion of the problem file formulation.

Domain File Design

The domain file was constructed to ensure valid robot actions, efficient product deliveries, and strict adherence to warehouse rules. The major components are outlined below.

Robot Movements Robots can move **up**, **down**, **left**, or **right** within the grid:

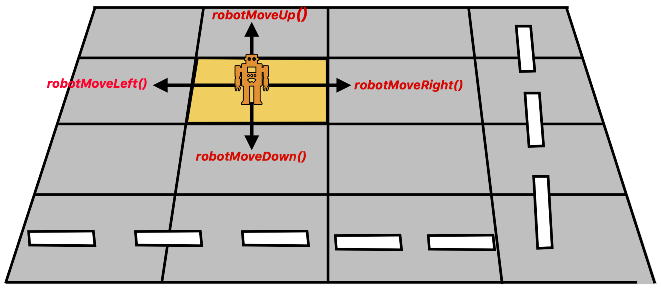


Figure 2: Robot movements visualised from current node.

- Moves: `robotMoveUp()`, `robotMoveDown()`, `robotMoveLeft()`, `robotMoveRight()`.

Shelf Handling Robots can **pick up** and **put down** shelves with the following actions:

- `pickUp()`, `putDown()`.

Product Delivery Robots must deliver products from shelves to the correct picking stations:

- Delivery action: `deliver()`.

Highway and Picking Station Constraints Constraints to prevent:

- Shelves being placed on highways or picking stations.
- Robots placing shelves at invalid nodes.

Action Preconditions and Effects

The domain file formally specifies the preconditions and effects for each action, following the commonsense law of inertia (i.e., properties remain unchanged unless explicitly altered).

1. Robot Movement

- Preconditions: Robots cannot move out of grid bounds.
- Effects: Update robot's position to the new node after a valid movement.

These movement rules ensure robots don't perform illegal moves, like stepping outside the warehouse. Updating robot positions after valid moves gave me a consistent, realistic simulation of robot navigation in a bounded space.

2. Picking Up a Shelf

- Preconditions: Robot and shelf must be at the same node.
- Effects: The shelf becomes attached to the robot.

Robots can only pick up shelves if they're physically nearby, representing the need for proximity for interaction. This ensures realistic behavior and constraints on shelf handling actions by spatial relationships in the warehouse.

3. Putting Down a Shelf

- Preconditions: Robot must be carrying the shelf.
- Effects: The shelf is placed onto the node occupied by the robot.

The put-down rules ensure that a robot can only put down a shelf if it's currently carrying one, maintaining consistency between the robot's carried objects and the warehouse state for safe shelf placement and logical consistency.

4. Delivering a Product

- Preconditions: Robot must be at the target picking station and carrying the correct shelf.
- Effects: Update the order's product quantity to reflect the delivered amount.

The delivery rules ensure accurate order fulfillment by delivering orders to the correct location and shelf. This resulted in reliable order completion tracking and proper matching of robot actions with customer requirements.

Through a structured approach of separating the domain and problem files, carefully defining robot actions, and enforcing necessary constraints, I crafted a complete working domain compatible with the Clingo ASP solver. The design ensures valid autonomous robot behavior and efficient order fulfillment in the automated warehouse scenario.

Main Results and Analysis

The goal is to minimize the total number of actions and the overall time required to complete all deliveries. Five instances were provided for this project, and after testing the domain file on all instances, the following results were obtained.

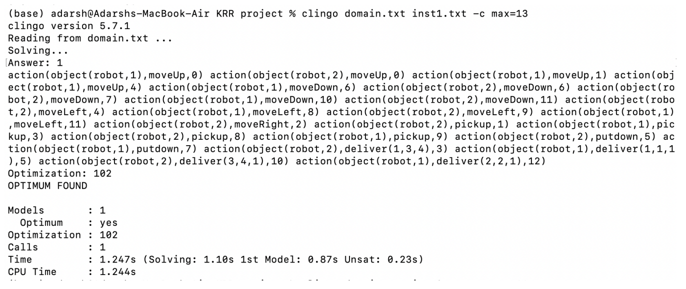
Instance No.	Min Steps Taken	CPU Time
1	13	1.244s
2	11	0.262s
3	7	0.050s
4	10	0.365s
5	6	0.035s

Table 1: Performance of domain file across provided instances

Clingo solvers were utilized, and the stable models reflected feasible plans to achieve the goal state from the initial configuration. The actions performed satisfied all logical constraints to deliver the products to the designated picking stations according to the order requirements. Analyzing the performance, it is observed that all instances completed with minimal steps, and the solver produced results in less than or close to one second for each instance. Testing was performed using a terminal on a MacBook Air equipped with an Apple M2 chip (10-core CPU, 8-core GPU).

Detailed Analysis of Instances

Since `inst1.asp` was dissected in earlier sections, a deeper analysis of its solution provides insights into the robot coordination strategy.



```
(base) adarsh@Adarshs-MacBook-Air KRR project % clingo domain.txt inst1.txt -c max=13
clingo version 5.7.1
Reading from domain.txt ...
Solving...
Answer: 1
action(object(robot,1),moveUp,0) action(object(robot,2),moveUp,0) action(object(robot,1),moveUp,1) action(obj
ect(robot,1),moveUp,4) action(object(robot,1),moveDown,6) action(object(robot,2),moveDown,6) action(object(ro
bot,2),moveDown,7) action(object(robot,1),moveDown,10) action(object(robot,2),moveDown,11) action(object(robo
t,2),moveLeft,4) action(object(robot,1),moveLeft,8) action(object(robot,2),moveLeft,9) action(object(robot,1)
,moveLeft,11) action(object(robot,2),moveRight,2) action(object(robot,2),pickup,1) action(object(robot,1),pic
kup,3) action(object(robot,2),pickup,8) action(object(robot,1),pickup,9) action(object(robot,2),putdown,5) ac
tion(object(robot,1),putdown,7) action(object(robot,2),deliver(1,3,4),3) action(object(robot,1),deliver(1,1,1
),5) action(object(robot,2),deliver(3,4,1),10) action(object(robot,1),deliver(2,2,1),12)
Optimization: 102
OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 102
Calls        : 1
Time         : 1.247s (Solving: 1.10s 1st Model: 0.87s Unsat: 0.23s)
CPU Time     : 1.244s
```

Figure 3: Robot 1 and 2 movement and shelf operations in Instance 1.

Coordination Strategy Analyzing the robot actions across all instances, a clear division of roles and efficient coordination strategies emerge. Robot 1 generally focuses on more straightforward vertical or lateral movements such as moving up, down, or left to pick up shelves and deliver products, often completing simpler delivery tasks efficiently. In contrast, Robot 2 typically adopts a more strategic approach, combining lateral and vertical movements, handling multiple product deliveries, and optimizing routes to minimize overall time and actions.

Specifically, in Instance 1, Robot 1 handled vertical movements to deliver Product 2, while Robot 2 delivered Products 1 and 3 with more dynamic movements. In Instance 2, Robot 1 focused on delivering Product 3, while Robot 2 successfully coordinated the deliveries of Products 1 and 2. Instance 3 demonstrated Robot 1 handling Product 4 through left and upward movements, and Robot 2 delivering Product 2. In Instance 4, Robot 1 delivered Product 2, while Robot 2 picked up and delivered Product 1 with intermediate shelf handling. Lastly, in Instance 5, Robot 1 efficiently delivered Product 1, and Robot 2 executed strategic moves to deliver Product 4.

Across all scenarios, the robots consistently adhered to the warehouse constraints, optimized their actions to minimize total time and steps, and effectively divided tasks based on movement simplicity versus strategic complexity. This demonstrates the robustness and flexibility of the logical domain framework in coordinating autonomous robots to fulfill dynamic warehouse delivery requirements.

Time	Robot 1 Action	Robot 2 Action
0	Moves up	Moves up
1	Moves up	PickUp Shelf-6
2	——	Moves right
3	PickUp Shelf-3	Deliver 4xProduct-1
4	Moves up	Moves left
5	——	Puts down a shelf
6	Moves down	Moves down
7	PutDown Shelf-3	Moves down
8	Moves left	PickUp Shelf-5
9	PickUp Shelf-4	Moves left
10	Moves down	Deliver 1xProduct-3
11	Moves left	Moves down
12	Deliver 1xProduct-2	——

Table 2: Timeline of actions for Robot 1 and Robot 2 in Instance 1 output

Conclusion

With the goal of efficiently coordinating multiple robots to fulfill dynamic delivery orders, I was able to develop and implement an automated warehouse domain using Answer Set Programming (ASP) with this project. Through the implementation of hard constraints, careful robot action definition, and splitting of the project into distinct domain and problem files, I was able to develop a working example that reduces the total time and action count in various warehouse scenarios which produced stable models that met every requirement and effectively delivered products to the assigned picking stations using Clingo solvers.

Based on how this project established I am confident that I demonstrated a firm understanding of declarative modeling, optimization techniques, and ASP ideas. I was able to successfully incorporate concepts such as the commonsense law of inertia, optimization goals, and choice rules into the warehouse management framework. The ability to run several instances and generate quick, reliable responses in a matter of seconds on a standard MacBook setup shows that the methodology is scalable and efficient for medium-sized problems. All things considered, I believe I achieved the goals I set out to achieve with this project: optimizing work completion in dynamic circumstances, ensuring safe and effective robot behavior, and simulating an automated warehouse using ASP.

Future Work

Although I am happy with the system's current features and design, I see plenty of opportunities to make it better and carry this project further into an industry scale. Based on my observations, future extensions include:

- **Industrial-scale extensions**:
  - Management of multiple interconnected warehouses.
  - Cross-functional robot operations across different warehouse zones.
  - Safe modeling of human-robot interaction.
  - Support for static and dynamic shelf restocking.
  - Incorporation of order deadlines and temporal delivery constraints.
- **Look-ahead planning strategies**: Enable robots to evaluate several future steps before committing to an action, helping them avoid bottlenecks or dead ends in dynamic layouts.
- **Heuristic-driven planning**: Prioritize actions or paths based on estimated cost or benefit to reduce solver workload, especially for larger and more complex grids.
- **Dynamic real-time re-planning**: Update plans during execution based on live feedback, such as blocked paths or urgent new orders.

Addressing these areas would significantly improve the system's robustness, adaptability, and scalability, bringing it closer to real-world large-scale warehouse automation standards.

Appendix: Full ASP Source Listings

A Execution Instructions

To run the automated warehouse planning scenario for Instance 1, use the **domain.txt** and **inst1.txt** files together with the Clingo solver. The execution command is:

```
clingo domain.txt inst1.txt -c max=13
```

This command instructs Clingo to load the domain and problem instance files, and sets the maximum planning horizon to 13 time steps for finding a valid and optimized plan.

B Domain.txt

```
%%%%%%%%%%%%%%%%%%%%%%%%%    ACTIONS    %%%%%%%%%%%%%%%%%%%%%%%%%
#const max=50.

%robotmoves in left, right, up, down
{robotMoveUp(R,T)}1 :- R=1..ROWS, nbots(ROWS), T=0..N,N=max-1.
{robotMoveDown(R,T)}1 :- R=1..ROWS, nbots(ROWS), T=0..N,N=max-1.
{robotMoveLeft(R,T)}1 :- R=1..ROWS, nbots(ROWS), T=0..N,N=max-1.
{robotMoveRight(R,T)}1 :- R=1..ROWS, nbots(ROWS), T=0..N,N=max-1.

%pickup and putdown
{pickUp(R,S,T):rack(S)}1:- nbots(ROWS), R=1..ROWS, T=0..N,N=max-1.
{putDown(R,S,T):rack(S)}1:- nbots(ROWS), R=1..ROWS, T=0..N,N=max-1.

%drop products
{drop(R,O,with(S,PR,DD),T):orderAt(O,object(spot,NODE),contains(PR,OQ),T),
↪  productOn(PR,object(rack,S),with(quant,PQ),T), DD=1..PQ}1:- R=1..ROWS,
↪  nbots(ROWS), T=0..N,N=max-1.

action(object(robot,R),moveUp,T) :- robotMoveUp(R,T).
action(object(robot,R),moveDown,T) :- robotMoveDown(R,T).
action(object(robot,R),moveLeft,T) :- robotMoveLeft(R,T).
action(object(robot,R),moveRight,T) :- robotMoveRight(R,T).
action(object(robot,R),pickup,T):-pickUp(R,_,T).
action(object(robot,R),putdown,T):-putDown(R,_,T).
action(object(robot,R),drop(O,PP,DD),T):-drop(R,O,with(S,PP,DD),T).

%%%%%%%%%%%%%%%%%%%%%%%%%    EFFECTS    %%%%%%%%%%%%%%%%%%%%%%%%%

% Robot move precondition and Effect
robotAt(R,object(spot,NEW_NODE),T+1) :- robotAt(R,object(spot,NODE),T),
↪  spotAt(NODE,pair(X,Y)), spotAt(NEW_NODE, pair(X-1,Y)), robotMoveUp(R,T).
robotAt(R,object(spot,NEW_NODE),T+1) :- robotAt(R,object(spot,NODE),T),
↪  spotAt(NODE,pair(X,Y)), spotAt(NEW_NODE, pair(X+1,Y)), robotMoveDown(R,T).
robotAt(R,object(spot,NEW_NODE),T+1) :- robotAt(R,object(spot,NODE),T),
↪  spotAt(NODE,pair(X,Y)), spotAt(NEW_NODE, pair(X,Y-1)), robotMoveLeft(R,T).
robotAt(R,object(spot,NEW_NODE),T+1) :- robotAt(R,object(spot,NODE),T),
↪  spotAt(NODE,pair(X,Y)), spotAt(NEW_NODE, pair(X,Y+1)), robotMoveRight(R,T).
```

```
% Lift rack Effect
rackOn(S,object(robot,RI),T+1):- pickUp(RI,S,T), rackOn(S,object(spot,NODE),T),
↪  robotAt(RI,object(spot,NODE),T).

% Drop rack Effect
rackOn(S,object(spot,NODE),T+1):- putDown(RI,S,T), rackOn(S,object(robot,RI),T),
↪  robotAt(RI,object(spot,NODE),T).

%Product drop Effect
orderAt(O,object(spot,NODE),contains(PR,OU-DD),T+1):- drop(R,O,with(S,PR,DD),T),
↪  orderAt(O,object(spot,NODE),contains(PR,OU),T).
productOn(PR,object(rack,S),with(quant,PQ-DD),T+1):- drop(R,O,with(S,PR,DD),T),
↪  productOn(PR,object(rack,S),with(quant,PQ),T).

%%%%%%%%%%%%%%%%%%%%%%%%%    CONSTRAINTS    %%%%%%%%%%%%%%%%%%%%%%%%%

% No simultaneous actions for a bot
:- action(object(robot,R),A,T), action(object(robot,R),AA,T), A!=AA.

% Cannot move beyond Warehouse
:- robotAt(RI,object(spot,NODE),T), spotAt(NODE,pair(X,Y)), X-1<1,
↪  robotMoveUp(RI,T).
:- robotAt(RI,object(spot,NODE),T), spotAt(NODE,pair(X,Y)), X+1>NC,
↪  numColumns(NC), robotMoveDown(RI,T).
:- robotAt(RI,object(spot,NODE),T), spotAt(NODE,pair(X,Y)), Y-1<1,
↪  robotMoveLeft(RI,T).
:- robotAt(RI,object(spot,NODE),T), spotAt(NODE,pair(X,Y)), Y+1>ROWS,
↪  numRows(ROWS), robotMoveRight(RI,T).

%While carrying a rack, a robot cannot move to a spot with a rack
:- robotAt(R,object(spot,NODE),T), rackOn(S,object(spot,NODE),T),
↪  rackOn(S2,object(robot,R),T), S != S2.

% One rack pickUp or putDown by one bot
:- 2{pickUp(R,S,T): robot(R)}, rack(S).
:- 2{putDown(R,S,T): robot(R)}, rack(S).

% No picking another rack if currently pickedUp
:- pickUp(RI,S1,T), rackOn(S2,object(robot,RI),T).
:- pickUp(R1,S,T), rackOn(S,object(robot,R2),T).
:- pickUp(RI,S,T), rackOn(S,object(spot,NODE),T), not
↪  robotAt(RI,object(spot,NODE),T).
:- putDown(RI,S,T), not rackOn(S,object(robot,RI),T).

% A bot cannot putdown a rack on a noway
:- putDown(RI,S,T), robotAt(RI,object(spot,NODE),T), noway(NODE).

% A robot cannot put down a rack at a spot containing a picking station
:- putDown(RI,S,T), robotAt(RI,object(spot,NODE),T), pickingStationAt(_,NODE).

% drop at PickingStations
:- drop(R,O,with(_,PR,_),T), orderAt(O,object(spot,NODE),contains(PR,_),T), not
↪  robotAt(R,object(spot, NODE),T).

% drop only shelves containing product
:- drop(R,O,with(S,PR,_),T), productOn(PR,object(rack,S),with(quant,_),T), not
↪  rackOn(S,object(robot,R),T).
```

```
:- drop(R,O,with(S,PR,DD),T), orderAt(O,object(spot,NODE),contains(PR,OQ),T),
↪  DD>OQ.
:- drop(R,O,with(S,PR,DD),T), productOn(PR,object(rack,S),with(quant,PQ),T),
↪  DD>PQ.

% noway constraints
:- rackOn(S,object(spot,NODEI),_), noway(NODEI).
:- pickingStationAt(_,NODEI), noway(NODEI).

% One robot cannot be in 2 spots at once
:- 2{robotAt(R,object(spot,NODE),T):spot(NODE)}, robot(R), T=0..max.

% Spot is unique for bots
:- 2{robotAt(R,object(spot,NODE),T):robot(R)}, spot(NODE), T=0..max.

% Swapping prohibited
:- robotAt(R1,object(spot,NODE1),T), robotAt(R1,object(spot,NODE2),T+1),
↪  robotAt(R2,object(spot,NODE2),T), robotAt(R2,object(spot,NODE1),T+1),
↪  R1!=R2.

% No rack on 2 bots
:- 2{rackOn(S,object(robot,ROWS),T): robot(ROWS)}, rack(S), T=0..max.

% 1 bot cannot carry two shelves
:- 2{rackOn(S,object(robot,ROWS),T): rack(S)}, robot(ROWS), T=0..max.

% one spot/robot has only one rack
:- 2{rackOn(S,object(spot,NODE),T): spot(NODE)}, rack(S), T=0..max.
:- 2{rackOn(S,object(spot,NODE),T): rack(S)}, spot(NODE), T=0..max.
:- rackOn(S,object(spot,_),T), rackOn(S,object(robot,_),T).

%%%%%%%%%%%%%%%%%%%%%%%%%    Common sense law     %%%%%%%%%%%%%%%%%%%%%%%%%

robotAt(R,object(spot,NODE),T+1) :- robotAt(R,object(spot,NODE),T), not
↪  robotMoveUp(R,T), not robotMoveDown(R,T), not robotMoveLeft(R,T), not
↪  robotMoveRight(R,T), T<max.
rackOn(S,object(spot,NODE),T+1):-rackOn(S,object(spot,NODE),T), not
↪  pickUp(_,S,T), T<max.
rackOn(S,object(robot,RI),T+1):-rackOn(S,object(robot,RI),T), not
↪  putDown(RI,S,T), T<max.
orderAt(O,object(spot,NODE),contains(PR,OU),T+1):-
↪  orderAt(O,object(spot,NODE),contains(PR,OU),T),
↪  productOn(PR,object(rack,S),with(quant,PQ),T), not drop(_,O,with(S,PR,_),T),
↪  T<max.
productOn(PR,object(rack,S),with(quant,PQ),T+1):-
↪  productOn(PR,object(rack,S),with(quant,PQ),T), not drop(_,_,with(S,PR,_),T),
↪  T<max.


% Goal state
:- not orderAt(O,object(spot,_),contains(PR,0),max),
↪  orderAt(O,object(spot,_),contains(PR,_),0).

#minimize{T:action(O,A,T)}.
#minimize{1,O,A,T:action(O,A,T)}.
#show action/3.
```

C Inst1.txt

```
%%%%%%%%%%%%%%%%%%%%%%%%%    FACTS    %%%%%%%%%%%%%%%%%%%%%%%%%
nbots(2). % There are 2 robots
robot(1). robot(2). % Robot IDs
numShelves(6). % There are 6 shelves
rack(1). rack(2). rack(3). rack(4). rack(5). rack(6). % rack IDs
numColumns(4). % The grid has 4 columns
numRows(4). % The grid has 4 rows
numNodes(16). % There are 16 spots
spot(1). spot(2). spot(3). spot(4). spot(5). spot(6). spot(7). spot(8).
spot(9). spot(10). spot(11). spot(12). spot(13). spot(14). spot(15). spot(16).

%%%%%%%%%%%%%%%%%%%%%%%%%  GRID POSITIONS      %%%%%%%%%%%%%%%%%%%%%%%%%
spotAt(1,pair(1,1)). spotAt(2,pair(2,1)). spotAt(3,pair(3,1)).
↪  spotAt(4,pair(4,1)).
spotAt(5,pair(1,2)). spotAt(6,pair(2,2)). spotAt(7,pair(3,2)).
↪  spotAt(8,pair(4,2)).
spotAt(9,pair(1,3)). spotAt(10,pair(2,3)). spotAt(11,pair(3,3)).
↪  spotAt(12,pair(4,3)).
spotAt(13,pair(1,4)). spotAt(14,pair(2,4)). spotAt(15,pair(3,4)).
↪  spotAt(16,pair(4,4)).

%%%%%%%%%%%%%%%%%%%%%%%%%  HIGHWAYS  %%%%%%%%%%%%%%%%%%%%%%%%%
noway(4). noway(8). noway(12). noway(13). noway(14). noway(15). noway(16).

%%%%%%%%%%%%%%%%%%%%%%%%%  PICKING STATIONS  %%%%%%%%%%%%%%%%%%%%%%%%%
pickingStationAt(1,9). % Picking Station 1 is at spot 9
pickingStationAt(2,3). % Picking Station 2 is at spot 3

%%%%%%%%%%%%%%%%%%%%%%%%%    ROBOTS    %%%%%%%%%%%%%%%%%%%%%%%%%
robotAt(1,object(spot,12),0). % Robot 1 is at spot 12
robotAt(2,object(spot,6),0). % Robot 2 is at spot 6

%%%%%%%%%%%%%%%%%%%%%%%%%    RACKS    %%%%%%%%%%%%%%%%%%%%%%%%%
rackOn(1,object(spot,11),0). % rack 1 is at spot 11
rackOn(2,object(spot,2),0).  % rack 2 is at spot 2
rackOn(3,object(spot,10),0).  % rack 3 is at spot 10
rackOn(4,object(spot,6),0).  % rack 4 is at spot 6
rackOn(5,object(spot,7),0). % rack 5 is at spot 7
rackOn(6,object(spot,5),0).  % rack 6 is at spot 5

%%%%%%%%%%%%%%%%%%%%%%%%%    PRODUCTS    %%%%%%%%%%%%%%%%%%%%%%%%%
productOn(1,object(rack,3),with(quant,1),0).
productOn(2,object(rack,4),with(quant,1),0).
productOn(3,object(rack,6),with(quant,4),0).
productOn(4,object(rack,5),with(quant,1),0).
productOn(4,object(rack,6),with(quant,1),0).

%%%%%%%%%%%%%%%%%%%%%%%%%    ORDERS    %%%%%%%%%%%%%%%%%%%%%%%%%
% Order1: Deliver Product 1 (quant 1) to Picking Station 1
% Order1: Deliver Product 3 (quant 4) to Picking Station 1
orderAt(1,object(spot,9),contains(1,1),0).
orderAt(1,object(spot,9),contains(3,4),0).

% Order2: Deliver Product 2 (quant 1) to Picking Station 2
orderAt(2,object(spot,3),contains(2,1),0).

% Order3: Deliver Product 4 (quant 1) to Picking Station 2
orderAt(3,object(spot,3),contains(4,1),0).
```

D Inst2.txt

```
%%%%%%%%%%%%%%%%%%%%%%%%%    FACTS    %%%%%%%%%%%%%%%%%%%%%%%%%
nbots(2).                          % 2 robots
robot(1). robot(2).
numShelves(5).                     % 5 racks (shelves)
rack(1). rack(2). rack(3). rack(4). rack(5).
numColumns(4).
numRows(4).
numNodes(16).
spot(1..16).

%%%%%%%%%%%  GRID MAPPING (row-col pairs)  %%%%%%%%%%%
spotAt(1 ,pair(1,1)).  spotAt(2 ,pair(2,1)).  spotAt(3 ,pair(3,1)).  spotAt(4
↪  ,pair(4,1)).
spotAt(5 ,pair(1,2)).  spotAt(6 ,pair(2,2)).  spotAt(7 ,pair(3,2)).  spotAt(8
↪  ,pair(4,2)).
spotAt(9 ,pair(1,3)).  spotAt(10,pair(2,3)).  spotAt(11,pair(3,3)).
↪  spotAt(12,pair(4,3)).
spotAt(13,pair(1,4)).  spotAt(14,pair(2,4)).  spotAt(15,pair(3,4)).
↪  spotAt(16,pair(4,4)).

%%%%%%%%%%%  HIGHWAYS (blocked lanes)  %%%%%%%%%%%
noway(4). noway(8). noway(12). noway(13). noway(14). noway(15). noway(16).

%%%%%%%%%%%  PICKING STATIONS  %%%%%%%%%%%
pickingStationAt(1,9).    % PS-1 at spot 9  (row 1, col 3)
pickingStationAt(2,3).    % PS-2 at spot 3  (row 3, col 1)

%%%%%%%%%%%  INITIAL ROBOT LOCATIONS (time 0)  %%%%%%%%%%%
robotAt(1,object(spot,12),0).   % Robot-1 at spot 12 (4,3)
robotAt(2,object(spot,6),0).    % Robot-2 at spot 6  (2,2)

%%%%%%%%%%%  RACK (SHELF) LOCATIONS (time 0)  %%%%%%%%%%%
rackOn(1,object(spot,11),0).    % rack 1 at spot 11
rackOn(2,object(spot,2 ),0).    % rack 2 at spot 2
rackOn(3,object(spot,10),0).    % rack 3 at spot 10
rackOn(4,object(spot,6 ),0).    % rack 4 at spot 6
rackOn(5,object(spot,7 ),0).    % rack 5 at spot 7

%%%%%%%%%%%  PRODUCT ALLOCATIONS (time 0)  %%%%%%%%%%%
productOn(1,object(rack,3),with(quant,1),0).   % Product 1 (1 unit) on rack 3
productOn(2,object(rack,4),with(quant,1),0).   % Product 2 (1 unit) on rack 4
productOn(3,object(rack,5),with(quant,3),0).   % Product 3 (3 units) on rack 5

%%%%%%%%%%%  ORDERS TO FULFIL (time 0)  %%%%%%%%%%%
% Order 1 (picking station 1):  P1×1  and  P3×2
orderAt(1,object(spot,9),contains(1,1),0).
orderAt(1,object(spot,9),contains(3,2),0).

% Order 2 (picking station 2):  P2×1
orderAt(2,object(spot,3),contains(2,1),0).
```

E Inst3.txt

```
%%%%%%%%%%%%%%%%%%%%%%%%%    FACTS    %%%%%%%%%%%%%%%%%%%%%%%%%
nbots(2).
robot(1). robot(2).
numShelves(6).
rack(1..6).
numColumns(4).
numRows(4).
numNodes(16).
spot(1..16).

%%%%%%%%%%%  GRID MAPPING  %%%%%%%%%%%
spotAt(1 ,pair(1,1)).  spotAt(2 ,pair(2,1)).  spotAt(3 ,pair(3,1)).  spotAt(4
↪  ,pair(4,1)).
spotAt(5 ,pair(1,2)).  spotAt(6 ,pair(2,2)).  spotAt(7 ,pair(3,2)).  spotAt(8
↪  ,pair(4,2)).
spotAt(9 ,pair(1,3)).  spotAt(10,pair(2,3)).  spotAt(11,pair(3,3)).
↪  spotAt(12,pair(4,3)).
spotAt(13,pair(1,4)).  spotAt(14,pair(2,4)).  spotAt(15,pair(3,4)).
↪  spotAt(16,pair(4,4)).

%%%%%%%%%%%  HIGHWAYS  %%%%%%%%%%%
noway(4).  noway(8).  noway(12).
noway(13). noway(14). noway(15). noway(16).

%%%%%%%%%%%  PICKING STATIONS  %%%%%%%%%%%
% Only one picking-station (ID 1) at spot 3  (pair 3,1)
pickingStationAt(1,3).

%%%%%%%%%%%  INITIAL ROBOT LOCATIONS (t = 0)  %%%%%%%%%%%
% Robot-1 at spot 12 (4,3) ; Robot-2 at spot 6 (2,2)
robotAt(1,object(spot,12),0).
robotAt(2,object(spot,6 ),0).

%%%%%%%%%%%  RACK LOCATIONS  %%%%%%%%%%%
rackOn(1,object(spot,11),0).
rackOn(2,object(spot,2 ),0).
rackOn(3,object(spot,10),0).
rackOn(4,object(spot,6 ),0).
rackOn(5,object(spot,7 ),0).
rackOn(6,object(spot,5 ),0).
```

```
%%%%%%%%%%%  PRODUCT ALLOCATIONS  %%%%%%%%%%%
% (derived from original pairs)
productOn(1,object(rack,3),with(quant,1),0).   % Product 1 on rack 3
productOn(2,object(rack,4),with(quant,1),0).   % Product 2 on rack 4
productOn(3,object(rack,6),with(quant,4),0).   % Product 3 on rack 6
productOn(4,object(rack,5),with(quant,1),0).   % Product 4 on rack 5
productOn(4,object(rack,6),with(quant,1),0).   % Product 4 also on rack 6

%%%%%%%%%%%  ORDERS  %%%%%%%%%%%
% Order 1 (PS-1): deliver Product 2 ×1
orderAt(1,object(spot,3),contains(2,1),0).

% Order 2 (PS-1): deliver Product 4 ×1
orderAt(2,object(spot,3),contains(4,1),0).
```

F Inst4.txt

```
%%%%%%%%%%%%%%%%%%%%%%%%%    FACTS    %%%%%%%%%%%%%%%%%%%%%%%%%
nbots(2).
robot(1). robot(2).
numShelves(6).
rack(1..6).
numColumns(4).
numRows(4).
numNodes(16).
spot(1..16).

%%%%%%%%%%%  GRID MAPPING  %%%%%%%%%%%
spotAt(1 ,pair(1,1)).  spotAt(2 ,pair(2,1)).  spotAt(3 ,pair(3,1)).  spotAt(4
↪  ,pair(4,1)).
spotAt(5 ,pair(1,2)).  spotAt(6 ,pair(2,2)).  spotAt(7 ,pair(3,2)).  spotAt(8
↪  ,pair(4,2)).
spotAt(9 ,pair(1,3)).  spotAt(10,pair(2,3)).  spotAt(11,pair(3,3)).
↪  spotAt(12,pair(4,3)).
spotAt(13,pair(1,4)).  spotAt(14,pair(2,4)).  spotAt(15,pair(3,4)).
↪  spotAt(16,pair(4,4)).

%%%%%%%%%%%  HIGHWAYS  %%%%%%%%%%%
noway(4).  noway(8).  noway(12).
noway(13). noway(14). noway(15). noway(16).

%%%%%%%%%%%  PICKING STATIONS  %%%%%%%%%%%
pickingStationAt(1,9).   % (row 1, col 3)
pickingStationAt(2,3).   % (row 3, col 1)

%%%%%%%%%%%  INITIAL ROBOT LOCATIONS (t = 0)  %%%%%%%%%%%
robotAt(1,object(spot,12),0).   % spot 12  (4,3)
robotAt(2,object(spot,6 ),0).   % spot 6   (2,2)

%%%%%%%%%%%  RACK LOCATIONS (t = 0)  %%%%%%%%%%%
rackOn(1,object(spot,11),0).
rackOn(2,object(spot,2 ),0).
rackOn(3,object(spot,10),0).
rackOn(4,object(spot,6 ),0).
rackOn(5,object(spot,7 ),0).
rackOn(6,object(spot,5 ),0).

%%%%%%%%%%%  PRODUCT ALLOCATIONS (t = 0)  %%%%%%%%%%%

productOn(1,object(rack,3),with(quant,1),0).
productOn(2,object(rack,4),with(quant,3),0).

%%%%%%%%%%%  ORDERS  %%%%%%%%%%%
% Order 1 (PS-1):  Product 1 ×1
orderAt(1,object(spot,9),contains(1,1),0).

% Order 2 (PS-2):  Product 2 ×1
orderAt(2,object(spot,3),contains(2,1),0).

% Order 3 (PS-2):  Product 2 ×2
orderAt(3,object(spot,3),contains(2,2),0).
```

G Inst5.txt

```
%%%%%%%%%%%%%%%%%%%%%%%%%    FACTS    %%%%%%%%%%%%%%%%%%%%%%%%%
nbots(2).
robot(1). robot(2).
numShelves(6).
rack(1..6).
numColumns(4).
numRows(4).
numNodes(16).
spot(1..16).

%%%%%%%%%%%  GRID MAPPING  %%%%%%%%%%%
spotAt(1 ,pair(1,1)).  spotAt(2 ,pair(2,1)).  spotAt(3 ,pair(3,1)).  spotAt(4
↪  ,pair(4,1)).
spotAt(5 ,pair(1,2)).  spotAt(6 ,pair(2,2)).  spotAt(7 ,pair(3,2)).  spotAt(8
↪  ,pair(4,2)).
spotAt(9 ,pair(1,3)).  spotAt(10,pair(2,3)).  spotAt(11,pair(3,3)).
↪  spotAt(12,pair(4,3)).
spotAt(13,pair(1,4)).  spotAt(14,pair(2,4)).  spotAt(15,pair(3,4)).
↪  spotAt(16,pair(4,4)).

%%%%%%%%%%%  HIGHWAYS  %%%%%%%%%%%
noway(4).  noway(8).  noway(12).
noway(13). noway(14). noway(15). noway(16).

%%%%%%%%%%%  PICKING STATIONS  %%%%%%%%%%%
pickingStationAt(1,9).      % Single picking-station at spot 9 (row 1,col 3)

%%%%%%%%%%%  INITIAL ROBOT LOCATIONS (t = 0)  %%%%%%%%%%%
robotAt(1,object(spot,12),0).   % Robot-1 at spot 12 (4,3)
robotAt(2,object(spot,6 ),0).   % Robot-2 at spot 6  (2,2)

%%%%%%%%%%%  RACK LOCATIONS (t = 0)  %%%%%%%%%%%
rackOn(1,object(spot,11),0).    % rack 1  → spot 11
rackOn(2,object(spot,2 ),0).    % rack 2  → spot 2
rackOn(3,object(spot,10),0).    % rack 3  → spot 10
rackOn(4,object(spot,6 ),0).    % rack 4  → spot 6   (shares node with Robot-2)
rackOn(5,object(spot,7 ),0).    % rack 5  → spot 7
rackOn(6,object(spot,5 ),0).    % rack 6  → spot 5

%%%%%%%%%%%  PRODUCT ALLOCATIONS (t = 0)  %%%%%%%%%%%
productOn(1,object(rack,3),with(quant,1),0).   % Product 1  (qty 1) on rack 3
productOn(2,object(rack,4),with(quant,1),0).   % Product 2  (qty 1) on rack 4
productOn(3,object(rack,6),with(quant,4),0).   % Product 3  (qty 4) on rack 6
productOn(4,object(rack,5),with(quant,1),0).   % Product 4  (qty 1) on rack 5
productOn(4,object(rack,6),with(quant,1),0).   % Product 4  (qty 1) also on rack
↪  6

%%%%%%%%%%%  ORDERS  %%%%%%%%%%%
% Order 1, Picking-Station 1 (spot 9):
orderAt(1,object(spot,9),contains(1,1),0).
orderAt(1,object(spot,9),contains(3,4),0).
```