Shortest-path algorithm as a tool for inner transportation optimization

109701056鍾沛臻(50%)、110701030陳羽潔(50%)

Outline

01

Introduction

02

Description of shortest path algorithms

03

Existing solutions

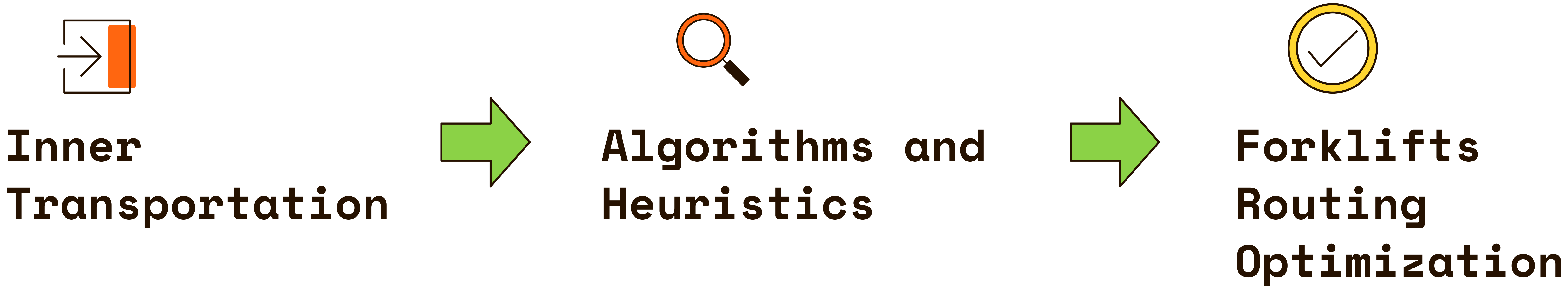
04

Proposed models

05

Conclusion and future work

Introduction



Inner Transportation

Algorithms and Heuristics

Forklifts Routing Optimization

Shortest Path Algorithms

Dijkstra algorithm

- single source SSP
- non-negative weights

A* algorithm

- single source SSP
- non-negative weights

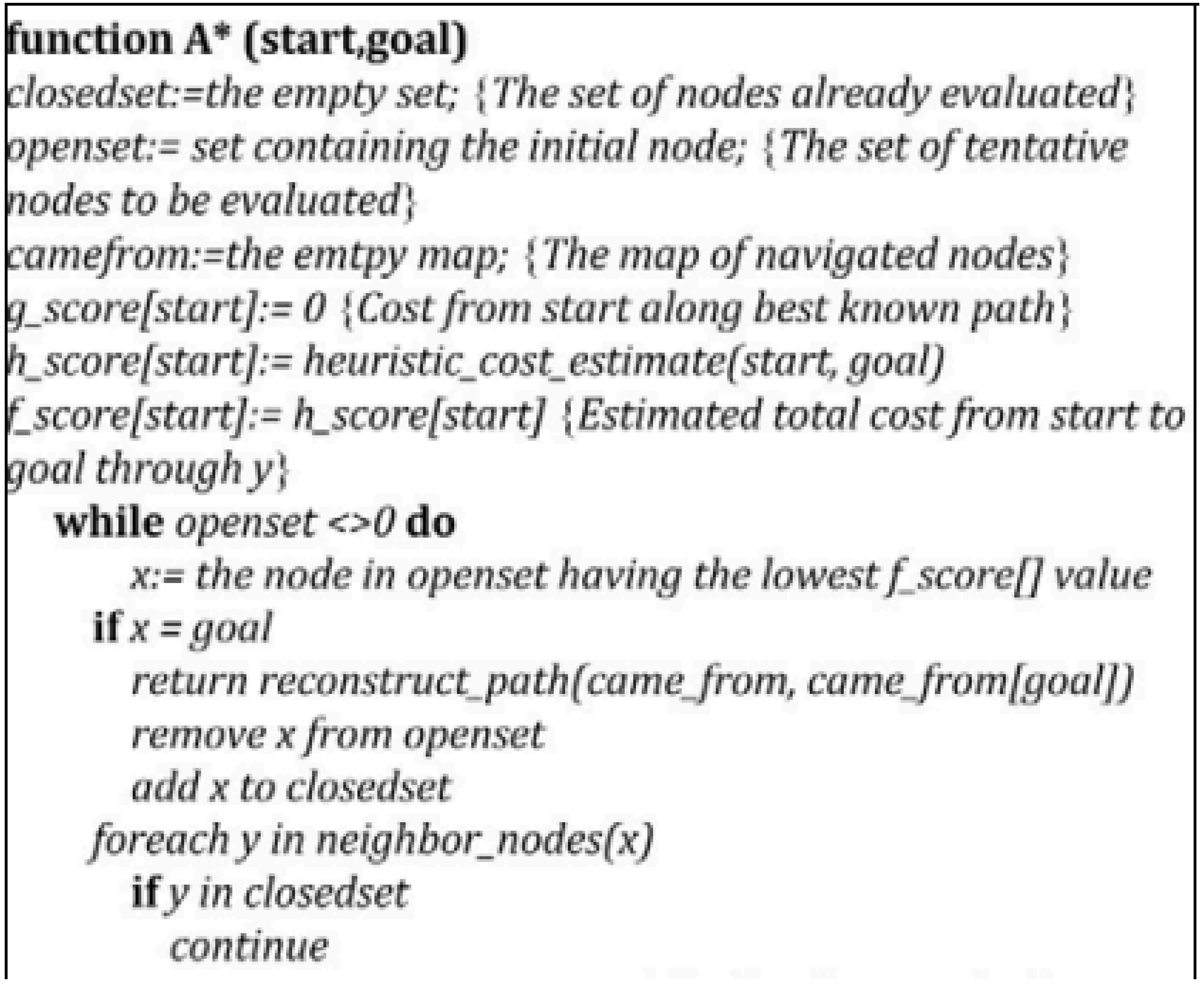
Bellman-Ford algortihm

- SP from a source to all vertex
- negative edge weights

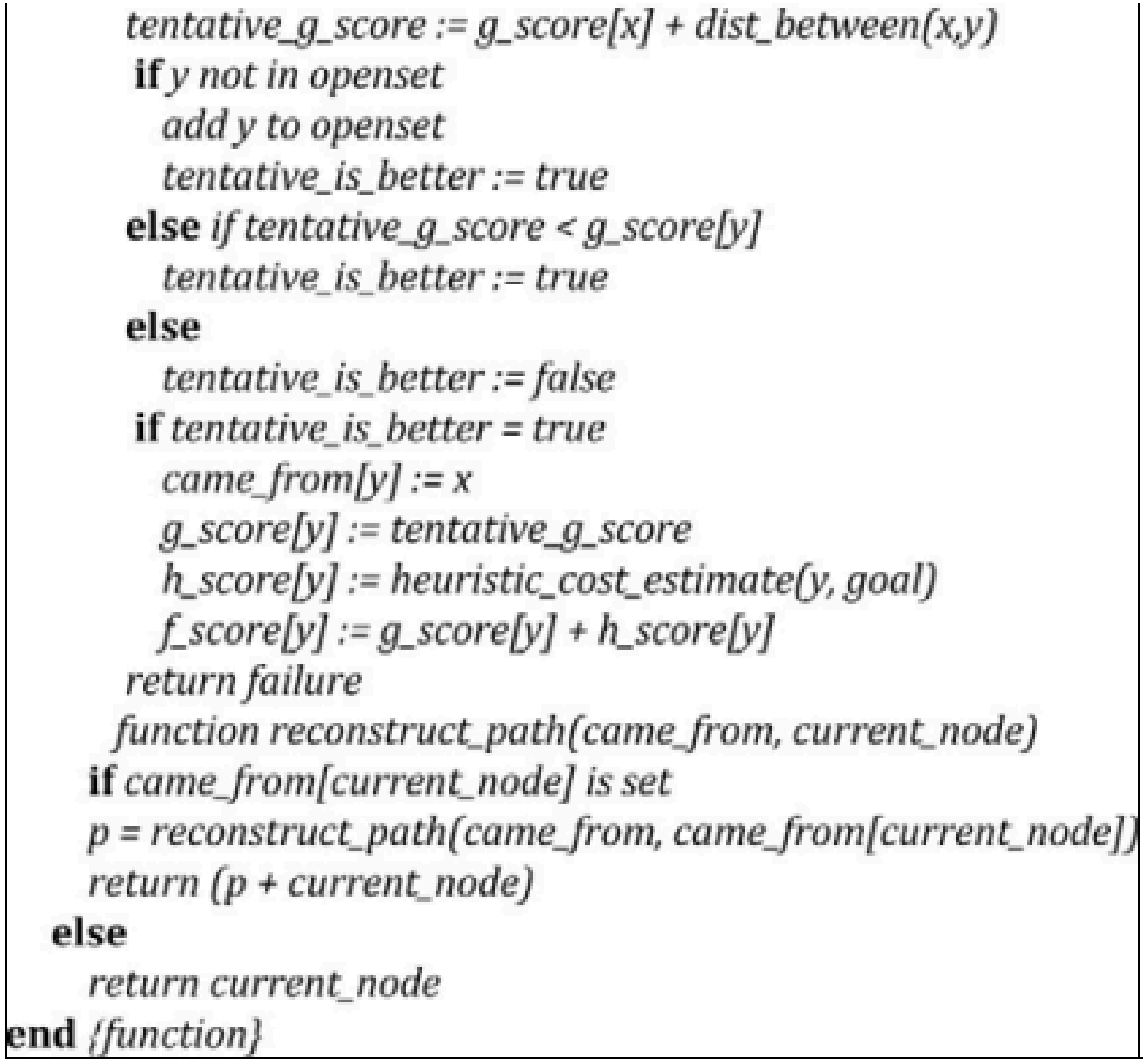
Floyd-Warshall algortihm

- SP between all vertices
- forbid negative cycles

Pseudo Code of A* algorithm



```
function A* (start,goal)
closedset:=the empty set; {The set of nodes already evaluated}
openset:= set containing the initial node; {The set of tentative nodes to be evaluated}
camefrom:=the emtpy map; {The map of navigated nodes}
g_score[start]:= 0 {Cost from start along best known path}
h_score[start]:= heuristic_cost_estimate(start, goal)
f_score[start]:= h_score[start] {Estimated total cost from start to goal through y}
  while openset <>0 do
      x:= the node in openset having the lowest f_score[] value
    if x = goal
      return reconstruct_path(came_from, came_from[goal])
      remove x from openset
      add x to closedset
    foreach y in neighbor_nodes(x)
      if y in closedset
        continue
```

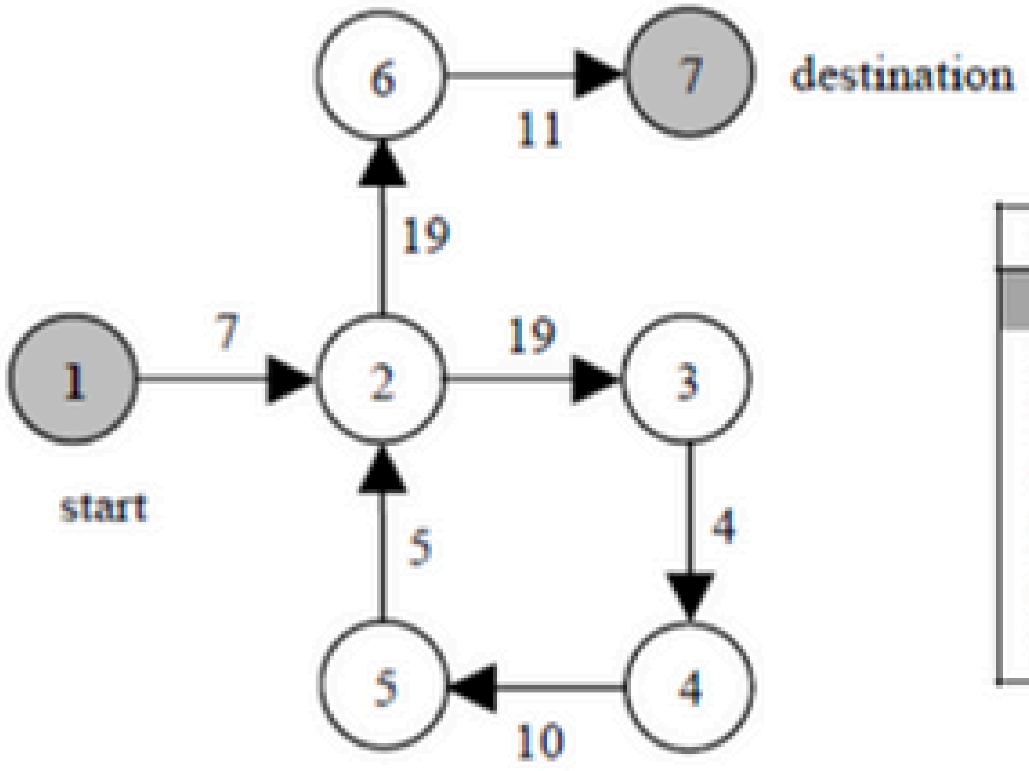


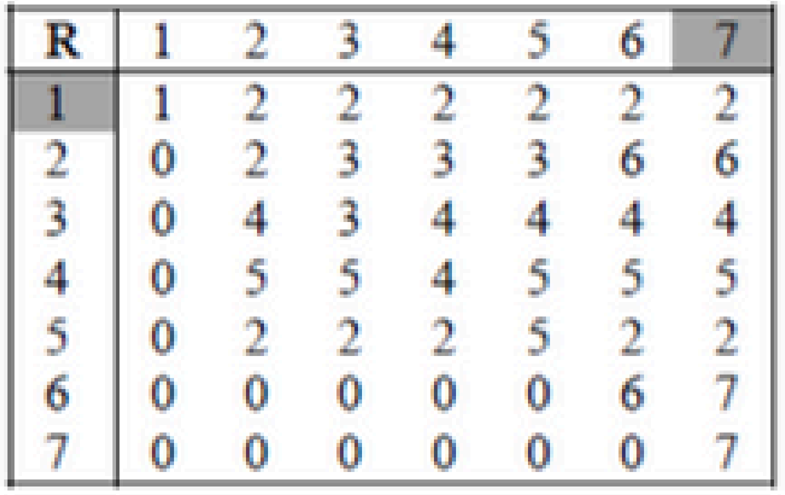
```
        tentative_g_score := g_score[x] + dist_between(x,y)
         if y not in openset
           add y to openset
           tentative_is_better := true
        else if tentative_g_score < g_score[y]
           tentative_is_better := true
        else
           tentative_is_better := false
         if tentative_is_better = true
           came_from[y] := x
           g_score[y] := tentative_g_score
           h_score[y] := heuristic_cost_estimate(y, goal)
           f_score[y] := g_score[y] + h_score[y]
        return failure
       function reconstruct_path(came_from, current_node)
      if came_from[current_node] is set
      p = reconstruct_path(came_from, came_from[current_node])
      return (p + current_node)
    else
      return current_node
end {function}
```

EXISTING SOLUTIONS

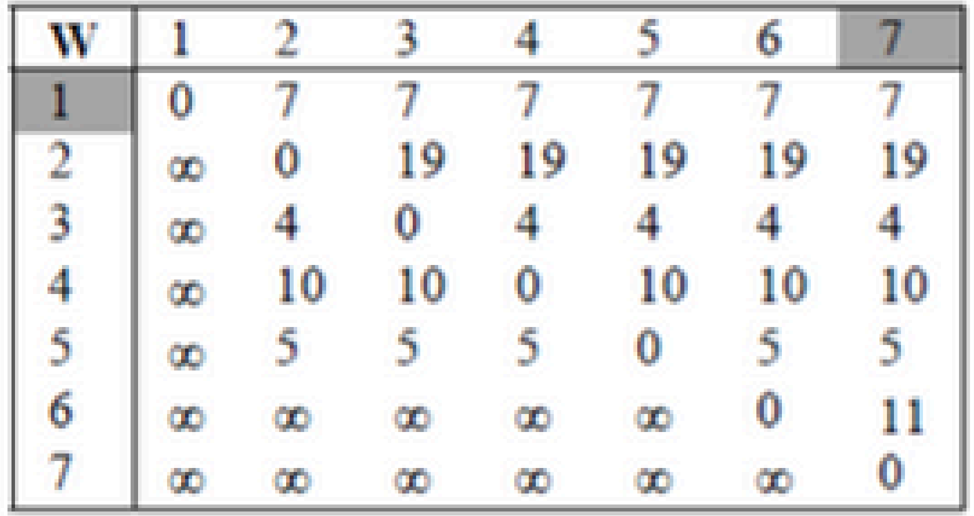
Hentschel

- Autonomous RTS-STILL robotic forklift truck
- Graph-based routing algorithm





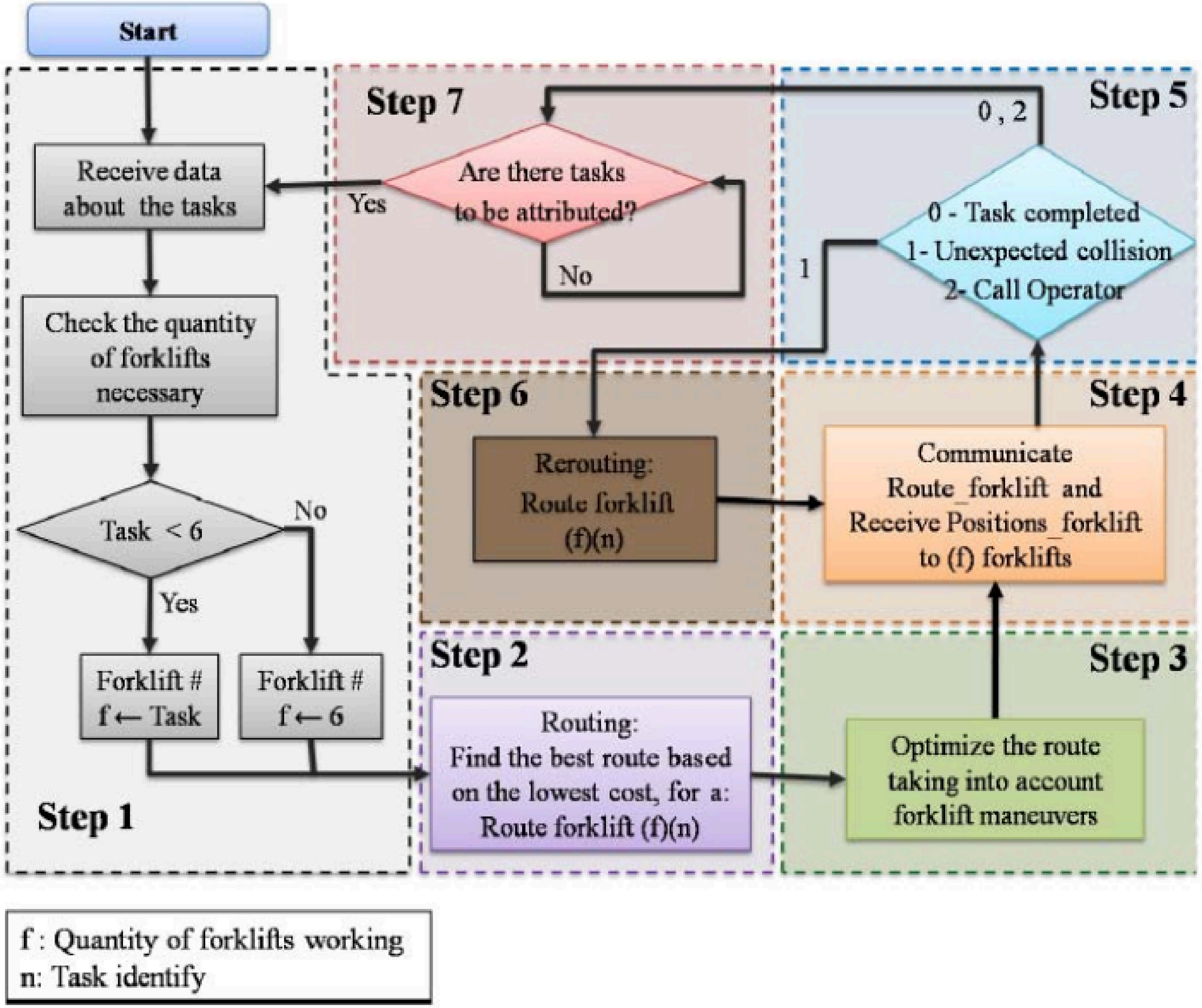
R	1	2	3	4	5	6	7
1	1	2	2	2	2	2	2
2	0	2	3	3	3	6	6
3	0	4	3	4	4	4	4
4	0	5	5	4	5	5	5
5	0	2	2	2	5	2	2
6	0	0	0	0	0	6	7
7	0	0	0	0	0	0	7



W	1	2	3	4	5	6	7
1	0	7	7	7	7	7	7
2	∞	0	19	19	19	19	19
3	∞	4	0	4	4	4	4
4	∞	10	10	0	10	10	10
5	∞	5	5	5	0	5	5
6	∞	∞	∞	∞	∞	0	11
7	∞	∞	∞	∞	∞	∞	0

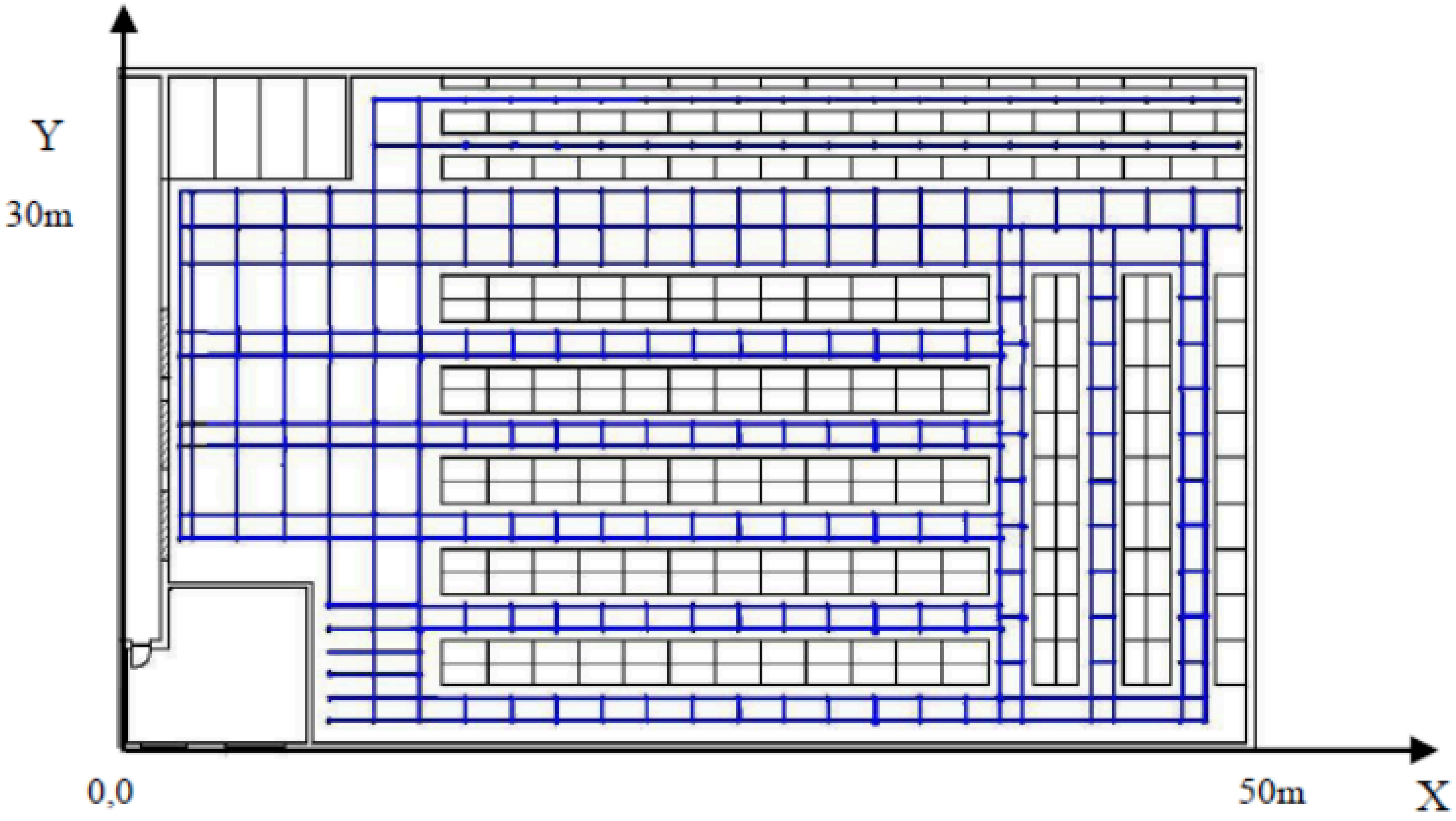
G = (V, E)
weight matrix W:costs for that particular path
route matrix R:storing the shortest paths in G

EXISTING SOLUTIONS


Start
Step 1
Receive data about the tasks
Check the quantity of forklifts necessary
Task < 6
Yes
No
Forklift # f← Task
Forklift # f← 6
Step 2
Routing: Find the best route based on the lowest cost, for a: Route forklift (f)(n)
Step 3
Optimize the route taking into account forklift maneuvers
Step 4
Communicate Route_forklift and Receive Positions_forklift to (f) forklifts
Step 5
0 - Task completed
1- Unexpected collision
2- Call Operator
0 , 2
1
Step 6
Rerouting: Route forklift (f)(n)
Step 7
Are there tasks to be attributed?
Yes
No
f : Quantity of forklifts working
n: Task identify

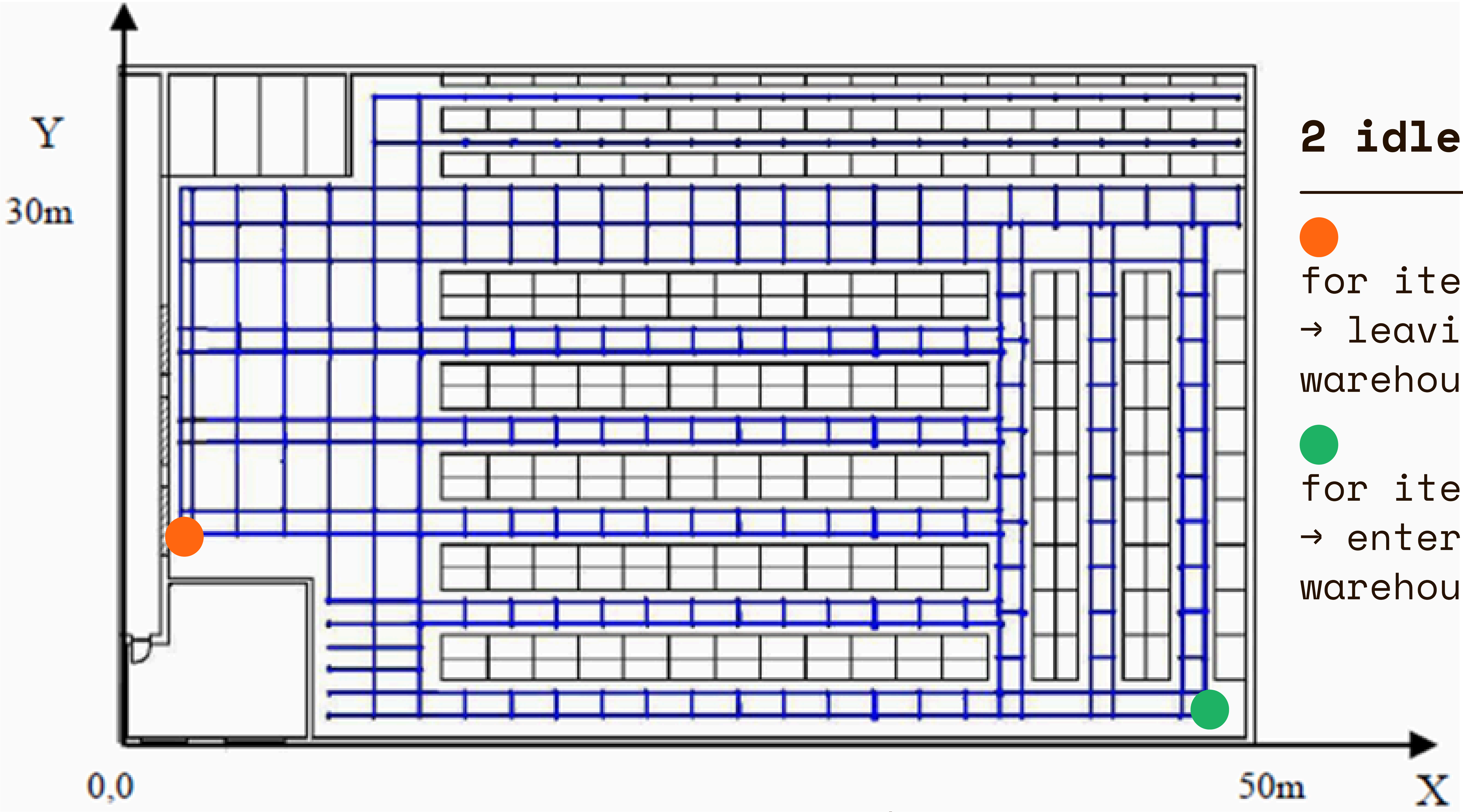
Vivaldini

- Intelligent warehouses routing system
- Conflict-free and optimized paths
- Software architecture



Proposed models

Optimize(determine) location of the new items

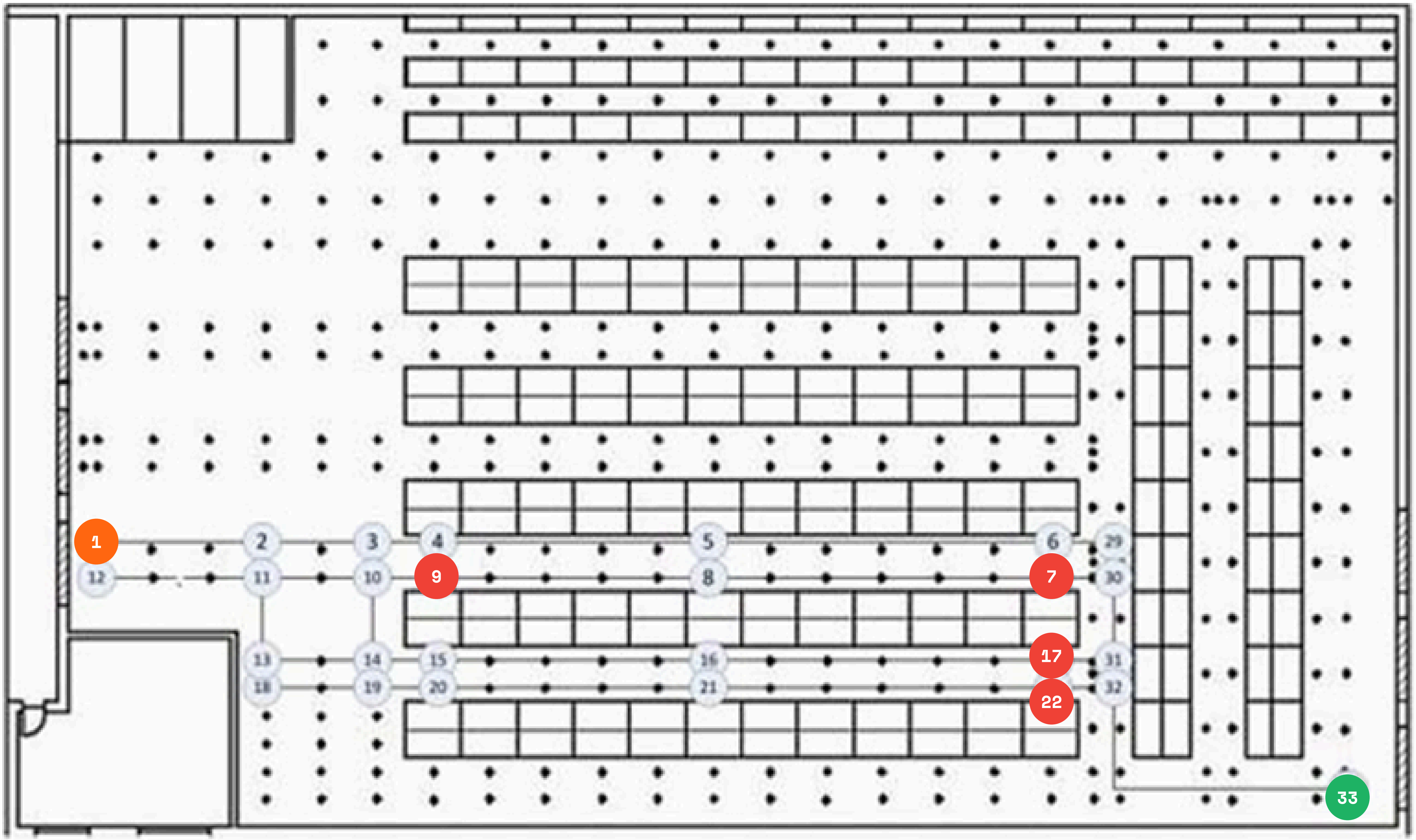


2 idle forklift

● for items loading → leaving the warehouse

● for items discharging → entering the warehouse

Proposed models



4 new items

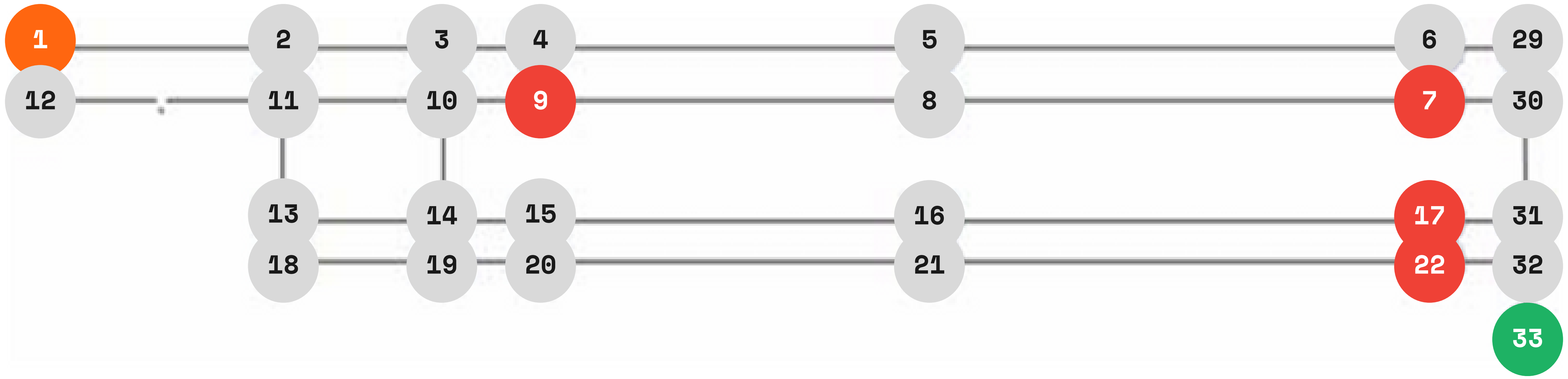
4 empty locations

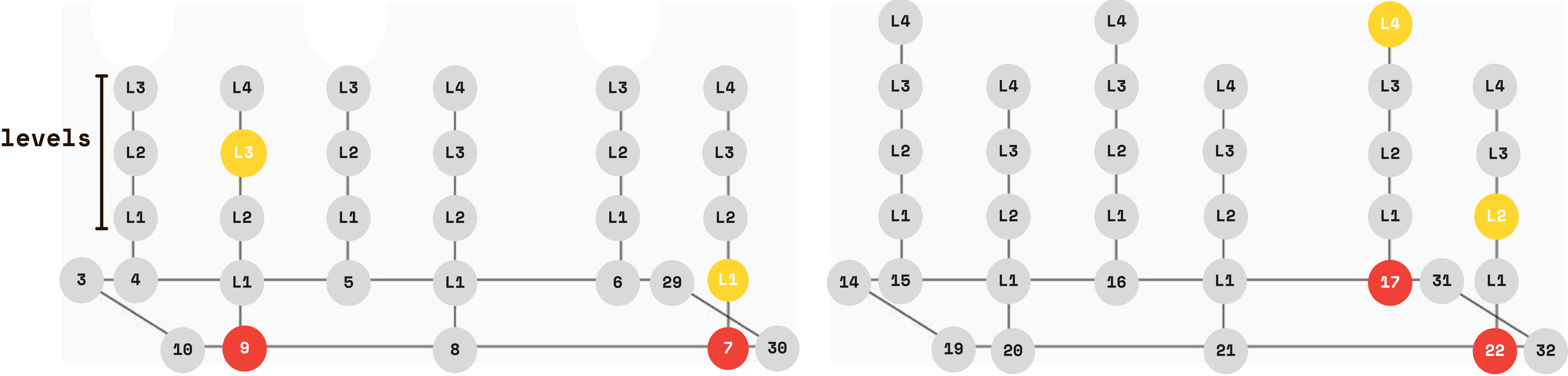
7 9 17 22

Three-dimensional (3D) warehouse storage locations

considering levels(shelves)

Proposed models



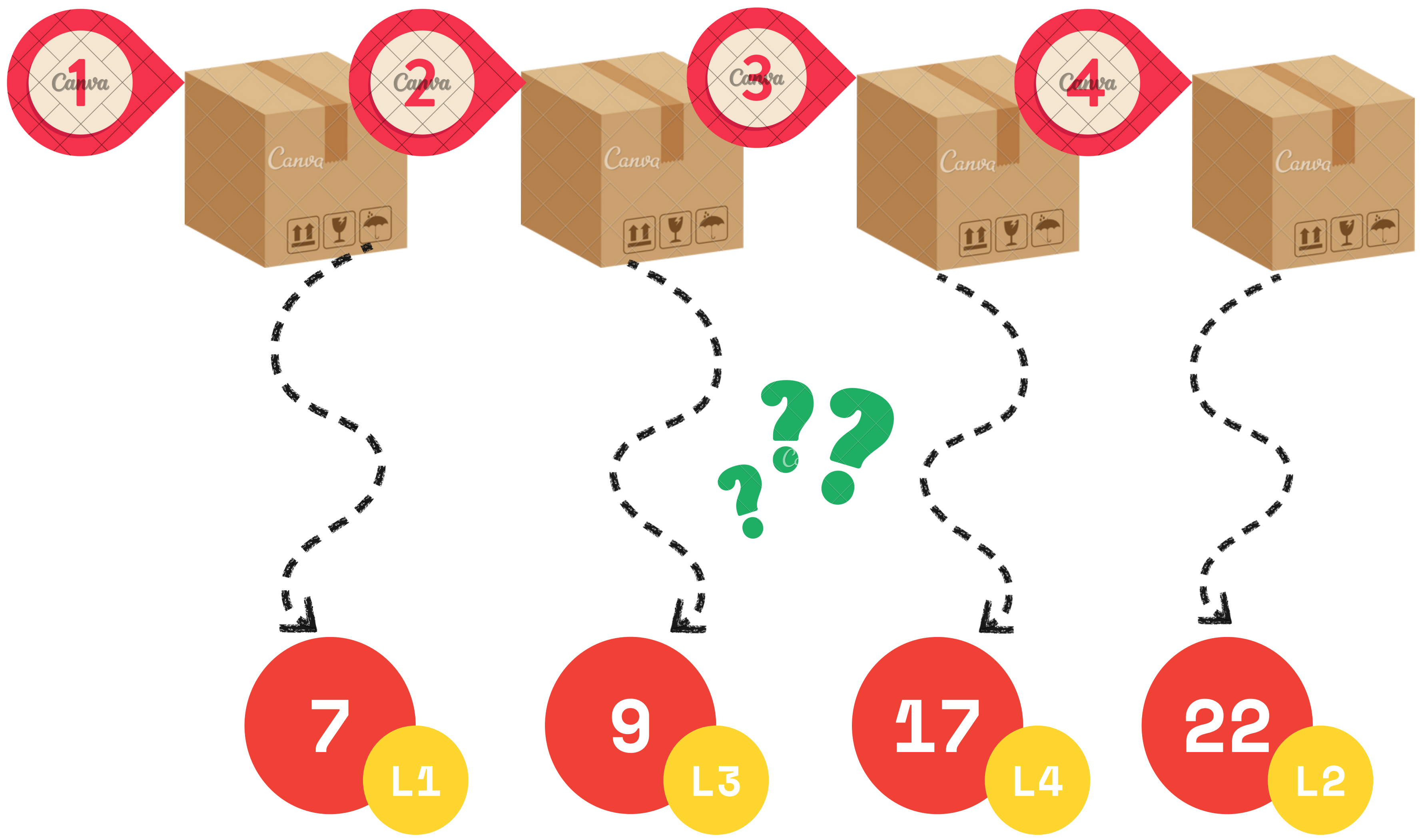


Proposed models

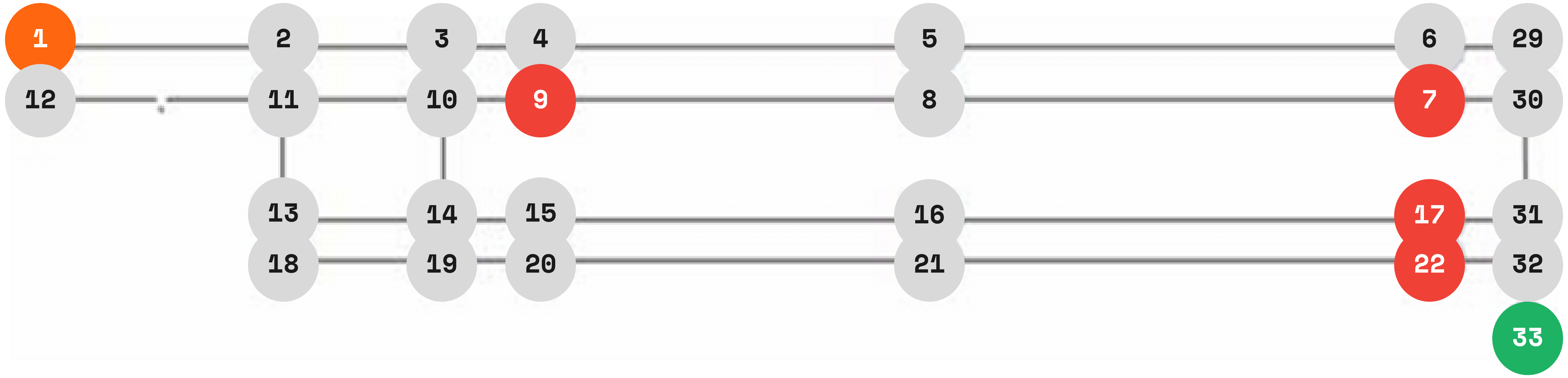
4 new items

4 empty locations

Higher priority
(used more frequently)



Proposed models



1 → 7

Dijkstra Algorithm
Find Shortest Path

33 → 7

Output Cost(Loading items)

+

Input Cost(Discharging items) = Total Cost

+

Cost for item lifting
on each level is 1

Proposed models

7

node: 7
output-cost: 24
input-cost: 15
lifting output-cost: 25
lifting input-cost: 16

total cost: 41

output path = 1 12 11 10 9 8 7
input path = 33 32 31 30 7

Lowest total cost
→ Put the item with the highest priority (used most frequently) on node 7

17

node: 17
output-cost: 27
input-cost: 12
lifting output-cost: 31
lifting input-cost: 16

total cost: 47

output path = 1 12 11 13 14 15 16 17
input path = 33 32 31 17

9

node: 9
output-cost: 12
input-cost: 27
lifting output-cost: 15
lifting input-cost: 30

total cost: 45

output path = 1 12 11 10 9
input path = 33 32 31 30 7 8 9

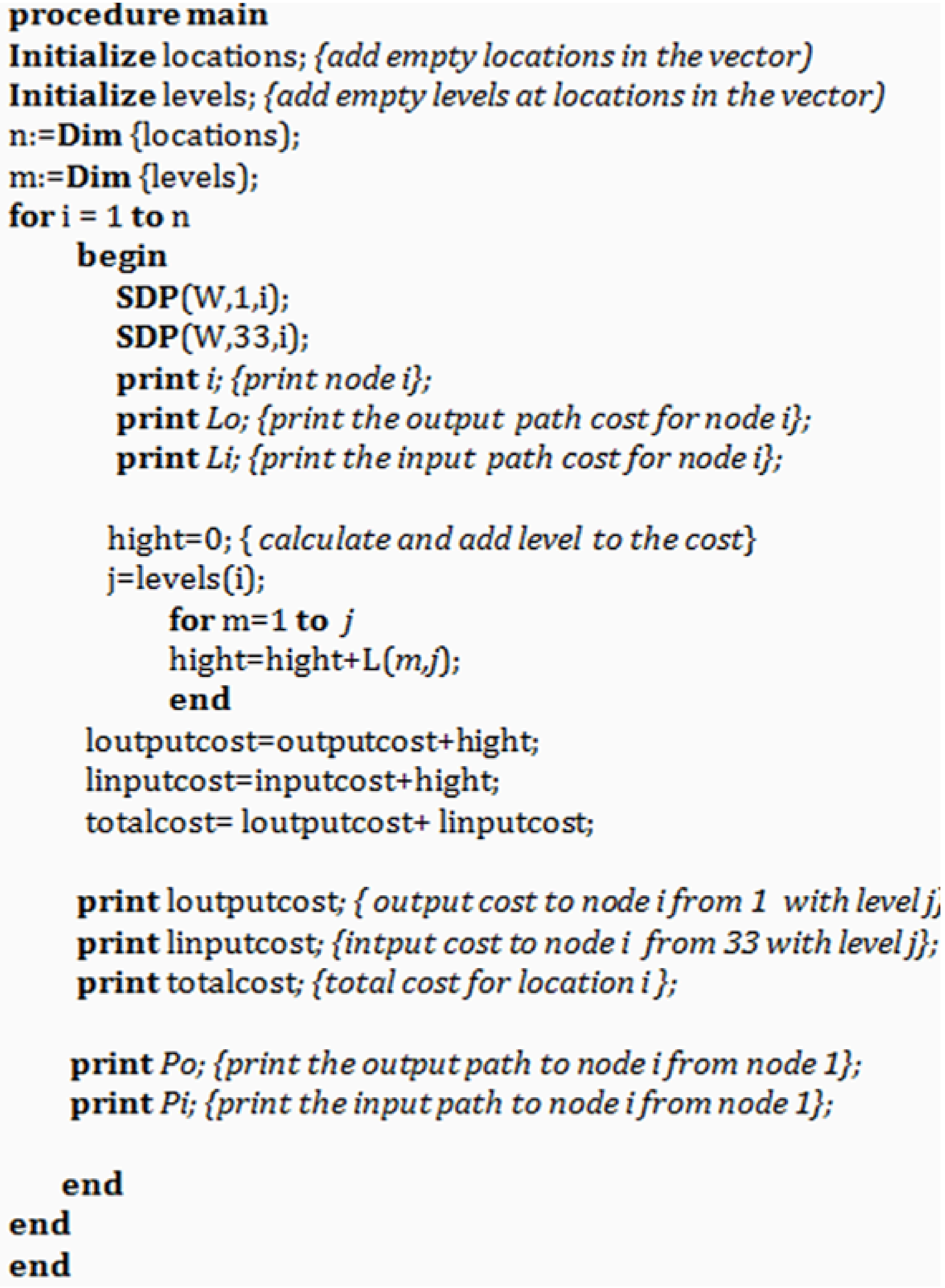
22

node: 22
output-cost: 28
input-cost: 11
lifting output-cost: 30
lifting input-cost: 13

total cost: 43

output path = 1 12 11 13 18 19 20 21 22
input path = 33 32 22

Pseudo Code for Item Location Optimization



```
procedure main
Initialize locations; {add empty locations in the vector)
Initialize levels; {add empty levels at locations in the vector)
n:=Dim {locations);
m:=Dim {levels);
for i = 1 to n
    begin
      SDP(W,1,i);
      SDP(W,33,i);
      print i; {print node i};
      print Lo; {print the output  path cost for node i};
      print Li; {print the input  path cost for node i};

     hight=0; { calculate and add level to the cost}
     j=levels(i);
         for m=1 to  j
         hight=hight+L(m,j);
         end
    loutputcost=outputcost+hight;
    linputcost=inputcost+hight;
    totalcost= loutputcost+ linputcost;

    print loutputcost; { output cost to node i from 1  with level j
    print linputcost; {intput cost to node i  from 33 with level j};
    print totalcost; {total cost for location i };

   print Po; {print the output path to node i from node 1};
   print Pi; {print the input path to node i from node 1};

   end
end
end
```

Conclusion and future work

Order picking cost
→55% of all operating costs

Can be applied to other types of vehicles and facilities

Improve business processes and productivity

Achieve costs reduced and automation of logistic processes

Thank you for your time!