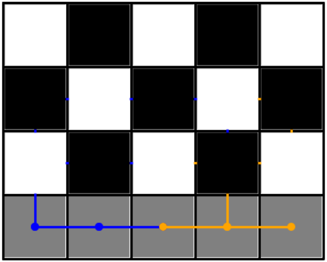
# The Floortile Problem

## DD2380 Artificial Intelligence

## Group Project



October 17th, 2017
KTH Royal Institute of Technology

*Group 22 :*

| | | |
|---|---|---|
| HEDSTRÖM Anna | 940307 – 3027 | annaheds@kth.se |
| LEGAT Antoine | 950311 – T297 | legat@kth.se |
| PICÓ ORISTRELL Sandra | 940531 – T268 | sandrapo@kth.se |
| VEGA RAMÍREZ David | 961202 – T031 | dvr@kth.se |

# Plan

# Introduction

## International Planning Competition

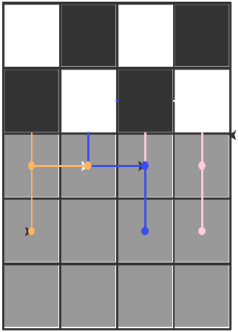
### Floortile domain

- N robots
- Deterministic environment
- Paint_up and Paint_down restriction
- Only paint with Black and White
- Robots can not stand on a painted tile

## How we solved the problem?

### PDDL

### Optimisation



# Plan

Introduction

PDDL

Optimisation

Discussion

Conclusion

# Formulation

Init $(\mathrm{Tile}(\mathrm{T}_1)(T_2) \wedge \ldots Tile(T_n) \wedge Tile(Rob_1)Robot(Rob_2) \wedge \ldots Robot(Rob_n) \wedge Color(White) \wedge Color(Black) \wedge Color(Color) \wedge (RobotAt(Robt_1, T_3) \wedge (RobotAt(Robt_2, T_2) \wedge (AvailableColor(White) \wedge (AvailableColor(Black) \wedge (RobotHas(Rob_1, White) \wedge (RobotHas(Rob_2, Black) \wedge (Clear(T_1)(Clear(T_2) \wedge (Clear(T_3) \wedge (Clear(T_n) \wedge (Up(T_3, T_1) \wedge (Up(T_4, T_2) \wedge (Up(T_5, T_n) \wedge (Down(T_1, T_3) \wedge (Down(T_2, T_4) \wedge (Down(T_5, T_n) \wedge (Right(T_2, T_1) \wedge (Right(T_2, T_3) \wedge (Right(T_5, T_n) \wedge (Left(T_1, T_2) \wedge (Left(T_3, T_2) \wedge (Left(T_5, T_n))$

Goal $(\mathrm{Painted}(\mathrm{T}_1, Black) \wedge (Painted(T_2, White)(T_1, \wedge Black) \wedge (Painted(T_2, White) \wedge (Painted(T_n, Color_n))$

Action(Change-Color)

PRECOND: $(\mathrm{RobotHas}(\mathrm{r}_1, c_1) \wedge AvailableColor(c_1))$

EFFECT: $(\mathrm{RobotHas}(\mathrm{r}_1, c_1) \neg RobotHas(r_1, c_2))$

Action(Paint-Up)

PRECOND: $(\mathrm{RobotHas}(\mathrm{r}_1, c_1) \wedge RobotAt(r_1, t_x) \wedge Up(t_y, t_x) \wedge Clear(t_y)$

EFFECT: $\mathrm{Clear}(\mathrm{t}_y) \neg Painted(t_y)$

Action(Paint-Down)

PRECOND: $(\mathrm{RobotHas}(\mathrm{r}_1, c_1) \wedge RobotAt(r_1, t_x) \wedge Down(t_y, t_x) \wedge Clear(t_y)$

EFFECT: $\mathrm{Clear}(\mathrm{t}_y) \neg Painted(t_y)$

Action(Up)

PRECOND: $\mathrm{RobotAt}(\mathrm{r}_1, t_x) \wedge Up(ty, t_x) \wedge Clear(t_y)$

# Formulation

EFFECT: RobotAt($r_1, t_y$)$\neg RobotAt(r_1, t_x) \wedge Clear(t_x) \neg Clear(t_y) \neg Painted(t_y)$

Action(Down)

PRECOND: RobotAt($r_1, t_x$) $\wedge Down(t_y, t_x) \wedge Clear(t_y)$

EFFECT: RobotAt$r_1 r, t_y$)$\neg RobotAt(r_1, t_x) \wedge Clear(t_x) \neg Clear(t_y) \neg Painted(t_y)$

Action(Right)

PRECOND: RobotAt($2_1, t_x$) $\wedge Right(t_y, t_x) \wedge Clear(t_y)$

EFFECT: RobotAt($r_1, t_y$)$\neg RobotAt(r_1, t_x) \wedge Clear(t_x) \neg Clear(t_y) notPainted(t_y)$

Action(Left)

PRECOND: RobotAt($r_{1,x}$ ) $\wedge Left(t_y, t_x) \wedge Clear(t_y)$

EFFECT: RobotAt($r_1, t_y$)$\neg RobotAt(r_1, t_x) \wedge Clear(t_x) \neg Clear(t_y) \neg Painted(t_y)$

# Interpretation

| | | |
|---|---|---|
| 1 | 0 | 3 |
| 0 | 0 | 2 |
| 3 | 0 | 1 |
| 0 | 0 | 0 |

FIGURE – Robot representation

| Value | Use |
|---|---|
| 0 | The Cell is clear |
| 1 | Cell has been painted white |
| 2 | Cell has been painted black |
| 3 | Robot is in top of the cell |

TABLE – Meaning of the values in the matrix

# Interpretation

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 2 | 100 | 100 |

FIGURE – Board representation

| Position | Use |
|---|---|
| 1 | X coordinate |
| 2 | Y coordinate |
| 3 | Current paint |
| 4 | Remaining black paint |
| 5 | Remaining white paint |

TABLE – Meaning of the values in the vector

# Implementation

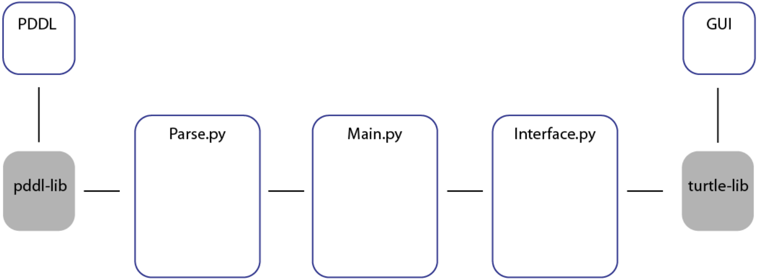



FIGURE – Python code schematic

# Implementation

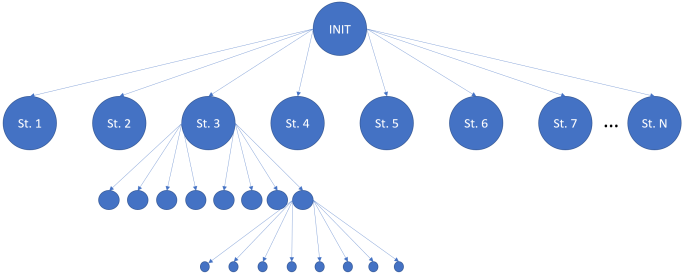



FIGURE – States traversal

# Implementation

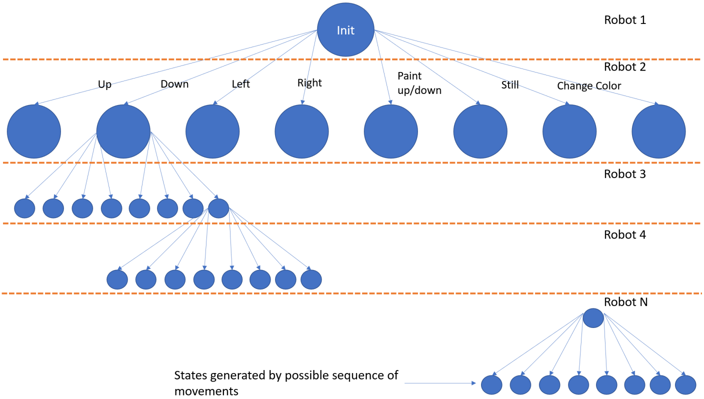



Figure – Obtaining possible states

# Implementation

**Algorithm 1** Planning algorithm

```
procedure PLANNING(ROBOTS,INITIALSTATE,TARGETSTATE)
    state its defined as [robots,initialState,sequenceMoves]
    q ← queue of states
    memory ← dictionary of states
    q ← push [robots,initialState,[] ]
    while:
    if q is empty then
        break
    current ← top(q)
    pop(q)
    if memory[current] == true) then
        continue
    if current[1] == targetState then
        return current
    memory[current] = true
    newPossibleStates ← getSequence(current[0], 0, current[1], targetState].
    For possible in newPossibleStates:
    q ← push [possible[0], possible[1], current[2].append(possible[2])
    goto For
    goto while.
    return null.
```

FIGURE – Planning algorithm

# Implementation

**Algorithm 2** getSequence algorithm

```
procedure GETSEQUENCE(ROBOTS,INDEX,STATE,TARGETSTATE,MOVEMENTS)
    if index == len(robots then
        resStates.append([robots,index,movements]
        return
    nextMovements ← getPossiblesFOC(robot[index],state,targetState)
    For next in nextMovements:
    getPossibles(robots,index+1,next[0],targetState,movements.append(next[1]))
    goto For
    if index == 0 then
        return resStates
```

FIGURE – getSequence

# Implementation

*(painted tile(3,1) black)* $\prec$ *(painted tile(2,1) white)* $\prec$ *(painted tile(1,1) black)*;
*(painted tile(3,2) white)* $\prec$ *(painted tile(2,2) black)* $\prec$ *(painted tile(1,2) white)*;
*(painted tile(3,3) black)* $\prec$ *(painted tile(2,3) white)* $\prec$ *(painted tile(1,3) black)*.

FIGURE – Forced Ordering Constraints

**Algorithm 3** getPossiblesFOC algorithm

```
procedure GETPOSSIBLES(ROBOT,STATE,TARGETSTATE)
    res = []
    movements = [up,down,left,right]
    For mov in movements:
    aux = tryMovement(mov,robot,state)
    if aux == NULL then
        res.append(aux)
    goto For
    if tryMovement(up,robot,state) and paintedColumn(robot,state) and state[robot[0:1]] == robot[2] then
        res.append(paintUp(robot,state))
    if tryMovement(downt,robot,state) and paintedColumn(robot,state) and state[robot[0:1]] == robot[2] then
        res.append(paintDown(robot,state))
    res.append(changeColor(robot,state))
    res.append(still(robot,state))
    return res
```

FIGURE – getPossiblesFOC

# Demo

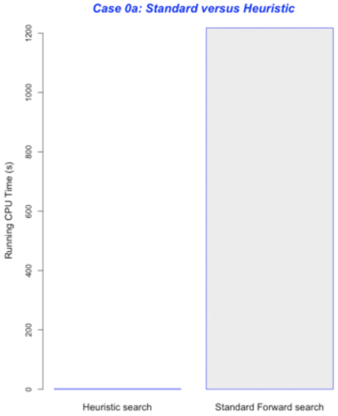
Demonstration

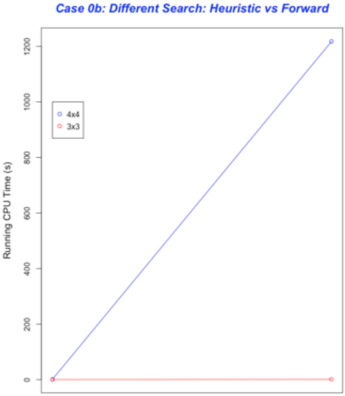
# Case studies

| | |
|---|---|
| Case 0 : Algorithms | • Algorithm without using FOC<br>• Algorithm using FOC |
| Case 1 : Number of robots | • 2 robots<br>• 3 robots<br>• 4 robots |
| Case 2: Pattern | • Target: all tiles painted<br>• Target: some tiles painted |
| Case 3: Size | • (3,3) • (3,4) • (5,4) • (5,5) • (5,6)<br>• (4,3) • (4,4) • (4,5) • (6,5) • (6,6) |

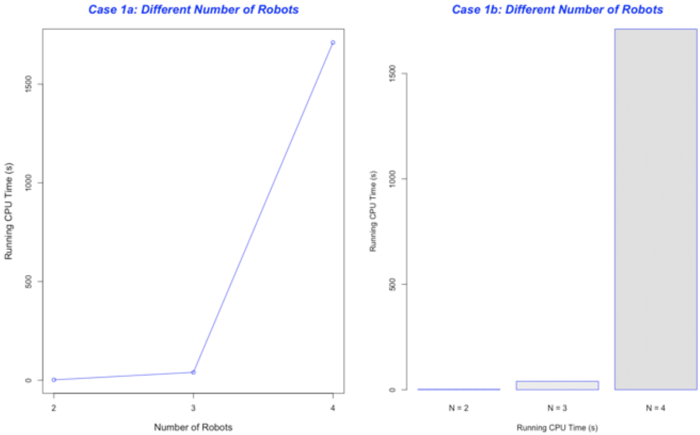
Figure – Different case studies in PDDL

# Case 0

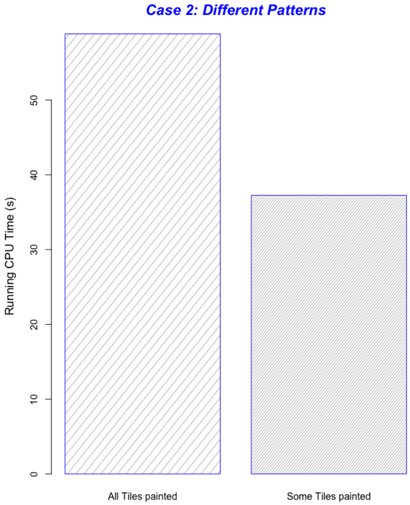

Case 0a: Standard versus Heuristic
Running CPU Time (s)
0
200
400
600
800
1000
1200
Heuristic search
Standard Forward search


Case 0b: Different Search: Heuristic vs Forward
Running CPU Time (s)
0
200
400
600
800
1000
1200
4x4
3x3

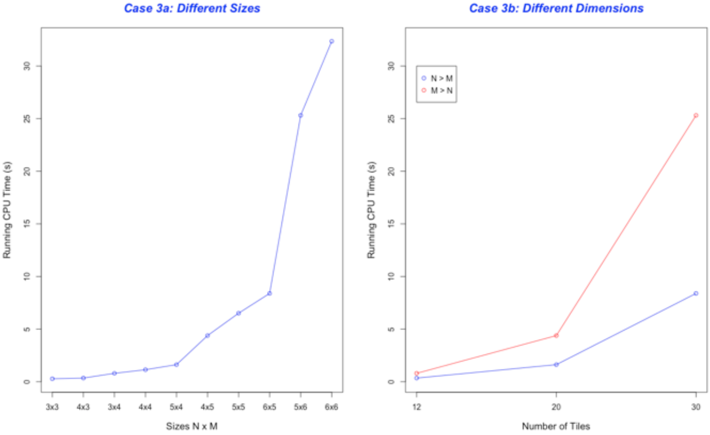
# Case 1


Case 1a: Different Number of Robots
Running CPU Time (s)
0
500
1000
1500
2
3
4
Number of Robots
Case 1b: Different Number of Robots
Running CPU Time (s)
0
500
1000
1500
N = 2
N = 3
N = 4
Running CPU Time (s)

# Case 2


Case 2: Different Patterns
Running CPU Time (s)
0
10
20
30
40
50
All Tiles painted
Some Tiles painted

# Case 3



Case 3a: Different Sizes
Running CPU Time (s)
0
5
10
15
20
25
30
3x3
4x3
3x4
4x4
5x4
4x5
5x5
6x5
5x6
6x6
Sizes N x M
Case 3b: Different Dimensions
Running CPU Time (s)
N > M
M > N
12
20
30
Number of Tiles

# Plan

# Formulation

- $t^*$ the time needed for complete board
- $\texttt{cell}(i, j, t)$ the state of cell $(i, j)$ at time $t$. $0 =$ not painted, $1 =$ painted
- $\texttt{painting}(i, j, t, r) = 1$ if robot $r$ is painting cell $(i, j)$ at time $t$
- State of the robots
    - $y(t, r)$ the vertical position of robot $r$ at time $t$
    - $x(t, r)$ the horizontal position of robot $r$ at time $t$
    - $\texttt{color}(t, r)$ the current color of robot $r$ at time $t$. $1 =$ white, $0 =$ black
    - $\texttt{stock0}(t, r)$ the current stock of black paint of robot $r$ at time $t$
    - $\texttt{stock1}(t, r)$ the current stock of white paint of robot $r$ at time $t$
- Actions of the robot
    - $\texttt{paint}(t, r) = 1$ if robot $r$ is painting at time $t$
    - $\texttt{move}(t, r) = 1$ if robot $r$ is moving at time $t$
    - $\texttt{switch}(t, r) = 1$ if robot $r$ is switching color at time $t$

# Formulation

$$
\begin{aligned}
\min \quad & t^* \\
\text{such that} \sum_{i,j} & \texttt{cell}(i,j,t^*) = nm && \text{Complete board} \\
& \texttt{paint}(t,r) + \texttt{move}(t,r) + \texttt{switch}(t,r) \leq 1 \quad \forall t,r && \text{One action at a time} \\
& \texttt{cell}(x(t,r), y(t,r), t) = 0 \quad \forall t,r && \text{Not stand on paint} \\
& x(t,r) = x(t,r') \Rightarrow y(t,r) \neq y(t,r') \quad \forall r, r' \neq r && \text{One robot per cell – 1} \\
& y(t,r) = y(t,r') \Rightarrow x(t,r) \neq x(t,r') \quad \forall r, r' \neq r && \text{One robot per cell – 2} \\
& \texttt{paint}(t,r) = 1 \Rightarrow \texttt{painting}(x(t,r), y(t,r)+1, t, r) \\
& \qquad + \texttt{painting}(x(t,r), y(t,r)-1, t, r) = 1 \quad \forall t,r && \text{Painting update} \\
& \texttt{paint}(t,r) = 1 \Rightarrow \sum_{i,j} \texttt{painting}(i,j,t,r) = 1 \quad \forall t,r && \text{Painting only one cell} \\
& \texttt{paint}(t,r) = 0 \Rightarrow \texttt{painting}(i,j,t,r) = 0 \quad \forall i,j,t,r && \text{Not painting} \\
& \texttt{cell}(i,j,t) = 0 \text{ and } \sum_{r} \texttt{painting}(i,j,t,r) \geq 1 \Rightarrow \texttt{cell}(i,j,t+1) = 1 \quad \forall i,j,t && \text{Cells update} \\
& \texttt{cell}(i,j,t) = 1 \Rightarrow \texttt{cell}(i,j,t+1) = 1 \quad \forall i,j,t && \text{Stay painted}
\end{aligned}
$$

# Formulation

$$\texttt{cell}(i,j,t)=0 \text{ and } \sum_r \texttt{painting}(i,j,t,r)=0 \Rightarrow \texttt{cell}(i,j,t+1)=0 \quad \forall i,j,t \qquad \text{Not painted}$$

$$\texttt{move}(t,r)=1 \Rightarrow |y(t+1,r)-y(t,r)|+|x(t+1,r)-x(t,r)|=1 \quad \forall t,r \qquad \text{Moving}$$

$$\texttt{move}(t,r)=0 \Rightarrow |y(t+1,r)-y(t,r)|+|x(t+1,r)-x(t,r)|=0 \quad \forall t,r \qquad \text{Not moving}$$

$$\texttt{switch}(t,r)=1 \Rightarrow |\texttt{color}(t+1,r)-\texttt{color}(t,r)|=1 \quad \forall t,r \qquad \text{Switching colors}$$

$$\texttt{switch}(t,r)=0 \Rightarrow \texttt{color}(t+1,r)=\texttt{color}(t,r) \quad \forall t,r \qquad \text{Not switching}$$

$$\begin{aligned}&\texttt{paint}(t,r)=1 \text{ and } \texttt{color}(t,r)=0 \Rightarrow \texttt{stock0}(t+1,r)=\texttt{stock0}(t,r)-1\\ &\qquad \text{and } \texttt{stock1}(t+1,r)=\texttt{stock1}(t,r) \quad \forall t,r\end{aligned} \qquad \text{Decrement black stock}$$

$$\begin{aligned}&\texttt{paint}(t,r)=1 \text{ and } \texttt{color}(t,r)=1 \Rightarrow \texttt{stock1}(t+1,r)=\texttt{stock1}(t,r)-1\\ &\qquad \text{and } \texttt{stock0}(t+1,r)=\texttt{stock0}(t,r) \quad \forall t,r\end{aligned} \qquad \text{Decrement white stock}$$

$$\begin{aligned}&\texttt{paint}(t,r)=0 \Rightarrow \texttt{stock0}(t+1,r)=\texttt{stock0}(t,r)\\ &\qquad \text{and } \texttt{stock0}(t+1,r)=\texttt{stock0}(t,r) \quad \forall t,r\end{aligned} \qquad \text{Constant stocks}$$

$$\begin{aligned}&\texttt{paint}(t,r)=1 \text{ and } \texttt{color}(t,r)=1 \Rightarrow \texttt{pattern}(x(t,r)+1,y(t,r))\\ &\qquad =\texttt{color}(t,r) \quad \forall t,r\end{aligned} \qquad \text{Respect pattern}$$

$$1 \le y(t,r) \le n, \quad 1 \le x(t,r) \le m \quad \forall t,r \qquad \text{Stay inside the board}$$

$$t^*, y(t,r), x(t,r), \texttt{stock0}(t,r), \texttt{stock1}(t,r) \in \mathbb{Z}^+ \quad \forall t,r \qquad \text{Integer variables}$$

$$\begin{aligned}&\texttt{cell}(i,j,t), \texttt{painting}(i,j,t,r), \texttt{color}(t,r), \texttt{paint}(t,r), \texttt{move}(t,r),\\ &\qquad \texttt{switch}(t,r) \in \{0,1\} \quad \forall i,j,t,r\end{aligned} \qquad \text{Binary variables}$$

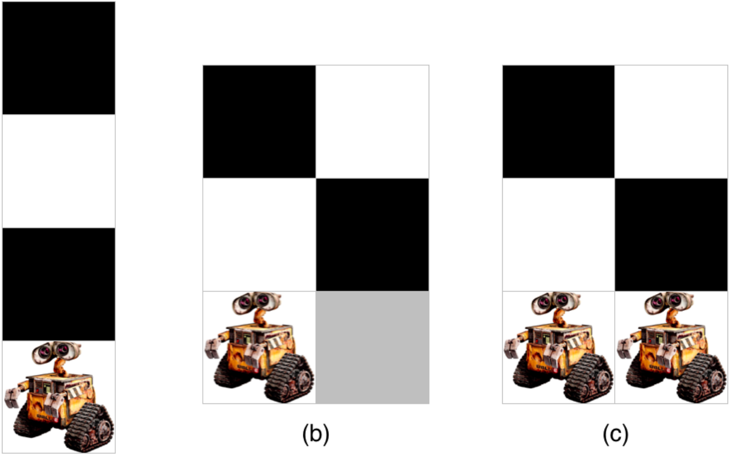
# Solver


*"IBM ILOG CPLEX CP Optimizer is a necessary and important complement to the optimization specialists' toolbox for solving real-world operational planning and scheduling problems – without a significant investment in R&D."*

# Demo

Demonstration

# Case studies



(a) (b) (c)

| | | |
|---|---|---|
| (a) 1.63608 | (b) 36.309 | (c) 117.117 |

TABLE – CPU times in seconds

# Plan

# Discussion

PDDL

Optimisation

# Plan

# Conclusion

Applications

Future work