Project Python

AspiR

Florent Valbon

Sorbonne Université

3/25/2020

Introduction

- Objective : design an optimize algorithm.
- On a grid read in a file, this algorithm gives the smaller number of displacements that n_{rob} robots can do to cover all the grid.

Classes

- **Room** : characterize the length and the width of a grid.
  - Method : **has_Wall()**.
- **AspiR** : model the problem.
  - Attributes : the grid, colors of robots, positions in the grid.
- **Robot** : characterize the color of a robot, and the problem associated.
  - Methods : **get_index()** (setter and getter), **can_move()**, **compute_move()** (move the robot).

Definitions and notations

- The problem is resolved in the file *main.py*.
- We use a method similar to trees. Even if trees are not implemented in Python, we use the type **list** to do that.
- The root refer to the initial configuration (positions of all robots at initial time).
- Each node refer to one configuration **BW**, **BS**, **BE**, **BN**, **RW**, **RS**, **RE** or **RN**.
- For a node, we will call **score** the number of different squares visited by all displacements associated to it and its parents nodes.
- Take a branch of the tree corresponds to add a new displacement in the grid (**BS** or **RS** etc.)
- A **level** of the tree correspond to a time when there is the same number of displacements in the grid. For instance :
  - level 2 : [**BW**, **BW**]… [**BW**, **RS**], … [**RN**,**RN**]
  - level 3 : [**BW**, **BW**, **BW**], [**BW**, **BW**, **BS**]…

Solution

- For each level of the tree :
  - The program look at all possible branches before take them, and for each possibilities, the **score** associated.
  - The algorithm select some branches among (we will precise how), and take them.
- When a branch is taken, the algorithm takes all branches selected in the level (the number of displacements in the grid remains the same) before go to an other level.
- The algorithm stopped the first time it find a node where the **score** is equal at the number of squared in the grid.

Strategies to optimize the program

- If all branches are taken, it's $\exp(k_{opt} \log(4 n_{rob}))$ combinations of displacements which are tested, before arrive to k_{opt} optimal displacements!

Strategies :

- Open the file only once, and stock its content in a list.
- Don't take the branches that involve a blocked robot (by a wall or an other robot).
- Don't take the branches that involve a round trip (for instance : **RS** after **RN**).
- Select only branches that lead to a better **score** (or at least equal) : this is controllable by the variable **precision** (the user can modify this).

Variable **precision**

- If **precision** $= 0$: the algorithm takes only branches that lead to a better **score** that the previous branch.
  - Warning : in a level, if k_{rest} is the number of displacements that rest before the optimal path is find, then for each branch rejected by this way, it's $\exp(k_{rest} \log(4 n_{rob}))$ combinations of displacement that will never be tested by the algorithm!
- If **precision** $= k_0 > 0$: the algorithm takes only branches that lead to a score that is at least equal to **better_score** - k_0 (this is less restrictive).
- Decrease **precision**, makes the algorithm faster, but solution can be not the best.
- Increase **precision**, makes the algorithm slower, but give it more chance to find an optimal solution.

Exemple

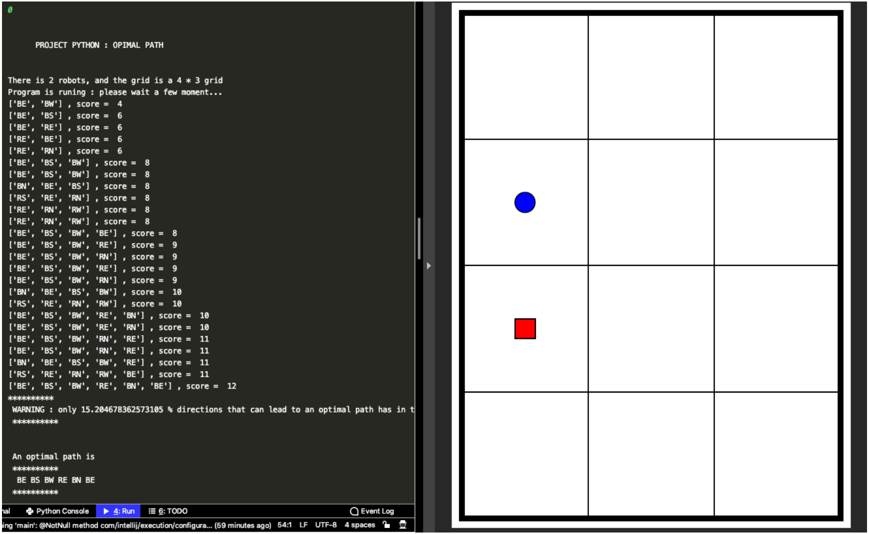

Figure 1: algorithm running for **precision** $= 0$

Results

Table 1: Results

Test_case	Results_obtained	Expected_results	Error	Execution_time	Precision
Case_Aspi_R_0	6	6	0	0.76 s	20
Case_Aspi_R_1	6	6	0	0.97 s	20
Case_Aspi_R_2	10	10	0	23.41 s	5
Case_Aspi_R_3	18	16	+2	24.1 s	3
Case_Aspi_R_4	15	13	+2	2.9 s	0
Case_Aspi_R_5	11	11	0	40 min	5
Case_Aspi_R_6	14	14	0	29 min	4
Case_Aspi_R_7	14	14	0	1h04	3

- For file **Case_Aspi_R_3** and **Case_Aspi_R_4** we should be increase the **precision** to have better results.