COMP203P: Scenario Week 4
Move-and-Tag Competition

University College London
Department of Computer Science
scenario@cs.ucl.ac.uk

20–24 February 2017

1 The Move-and-Tag Problem

Cleaning robots are becoming more and more popular nowadays. In order to maintain the cleanliness in a highly technological way, UCL has purchased a number of of cleaning robots of different models to be set up in different locations. The robots are highly intelligent: once woken up they, acting as a coordinated swarm, will take care of a location they are in, cleaning it spotless.

However, waking the robots up is not easy. As it has turned out, in each room there is only one dedicated robot that can be turned on automatically via a remote control. All other robots will have to be awakened by they peers, in other words, the very first robot will have to move around and "*tag*" its pals, so they can wake up and do the same to others, until the entire robot swarm wakes up and proceeds with the cleaning. Indeed, one robot can wake up several other robots subsequently, and, once activated, those robots can start waking more robots in parallel with the first one, therefore, reducing the overall "waking up time". For instance, Figure 1 shows a swarm of five robots, A–E at the moment when the first robot A wakes up (top-left square). It also demonstrates a possible waking sequence (top-right square), in which A first wakes up B and then E, whereas B, after it has woken up, wakes up C and D. Neither C, D or E awake anyone, instead, once tagged, they just remain at their initial positions until the the waking process is completed. The bottom of Figure 1 depicts the timeline of the waking sequence for the robots A–E, showing the temporal overlap between the periods of time when both A and B were active and moving.

Unfortunately, the robots are not located in empty rooms: instead they are surrounded by obstacles: tables, chairs, and piles of litter, which can be of any arbitrary form. Let us now take a look at a waking sequence in the presence of obstacles. The top-left part of Figure 2 shows a swarm of the same five robots A–E, but now two obstacles, O_1 and O_2 are introduced. Since the robots cannot move directly through the obstacles (sigh...), they will have to find ways around them. The top-right part of Figure 2 shows possible paths of robots A, B and C, moving to wake up their peers, once having been woken up themselves. Notice, that due to the presence of a large obstacle O_2 on A's way toward E, it appears to be more efficient, in terms of the overall time spent, for C to wake E, while B is tagging D. Importantly, the *shortest path* from B to C cannot be a straight line segment: instead B should change its direction, upon reaching the vertex X of O_1 on its way to C. Indeed, there might be other ways for B to reach C, or for A to tag C directly after. However, all these alternative options would have increased the total time spent by the swarm waking up.

Finding the *optimal* set of paths for each robot, which minimises the waking time (or, in more graphical terms, the "length" of the timeline) is what we are going to call *Move-and-Tag Problem* (MAT). The described challenge is a combination of a well-known *Robot Motion Planning* problem

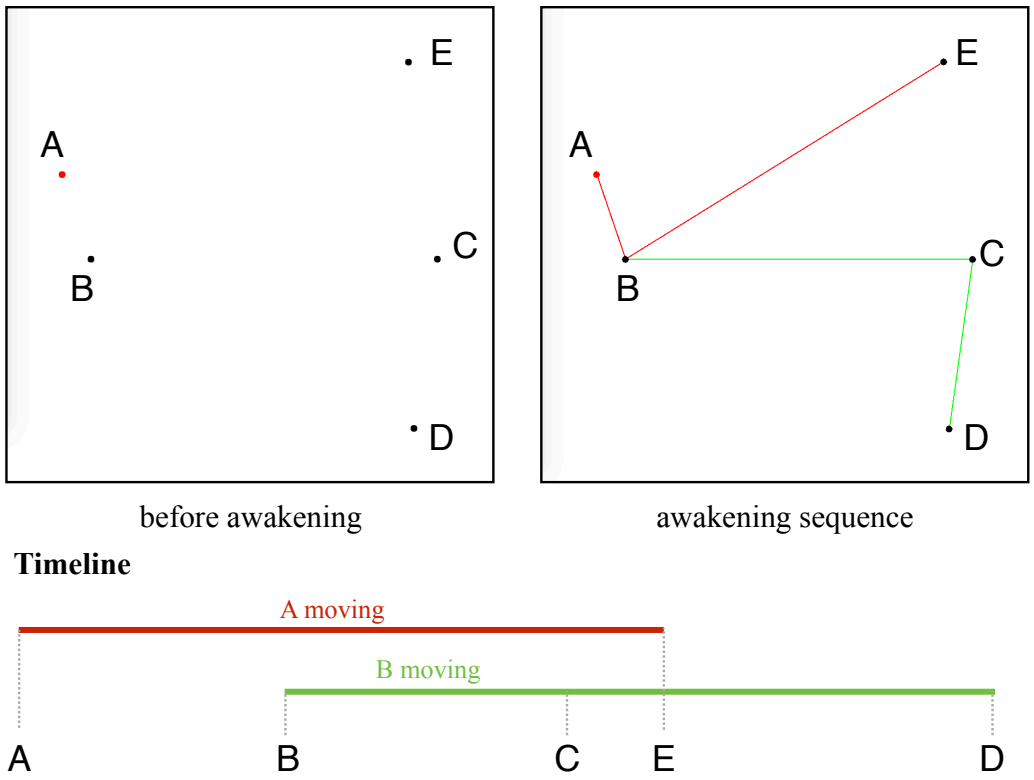

Figure 1: A simple waking sequence for five robots, starting from A.

(RMP) and less well-known *Freeze-Tag* problem (FT). In RMP, one should find an optimal way for a robot from a point A to a point B in the presence of multiple obstacles. In FT, a swarm of robots must wake up in the shortest possible time, starting from one specified robots.

This scenario week is dedicated to solving different instances of the *Move-and-Tag Problem*. As almost any optimization problem, MAT is NP-hard, meaning that computing the *best* possible solution for large instances will require (to the best of the humanity's knowledge to date) an algorithm of *exponential* complexity (you should remember what it is from the last year Theory II module), which might take days, months or even years of real time to terminate. Your main goal for this week is to design an algorithm, which finds MAT solutions that are, if not the best, are at least *valid*:

1. Robots' paths may not cross the obstacles (but can touch their boundaries);
2. *All* robots in the swarm should be "tagged" by the end;
3. The may be no "cycles" in the awakening sequences. For example, a robot B cannot be woken up by a robot C, that had been woken up by B.

The remainder of this document explains the details of each of the tasks of the scenario and describes the grading system (Section 2). Section 3 lists some useful links, and Section 4 provides answers to frequently asked questions.

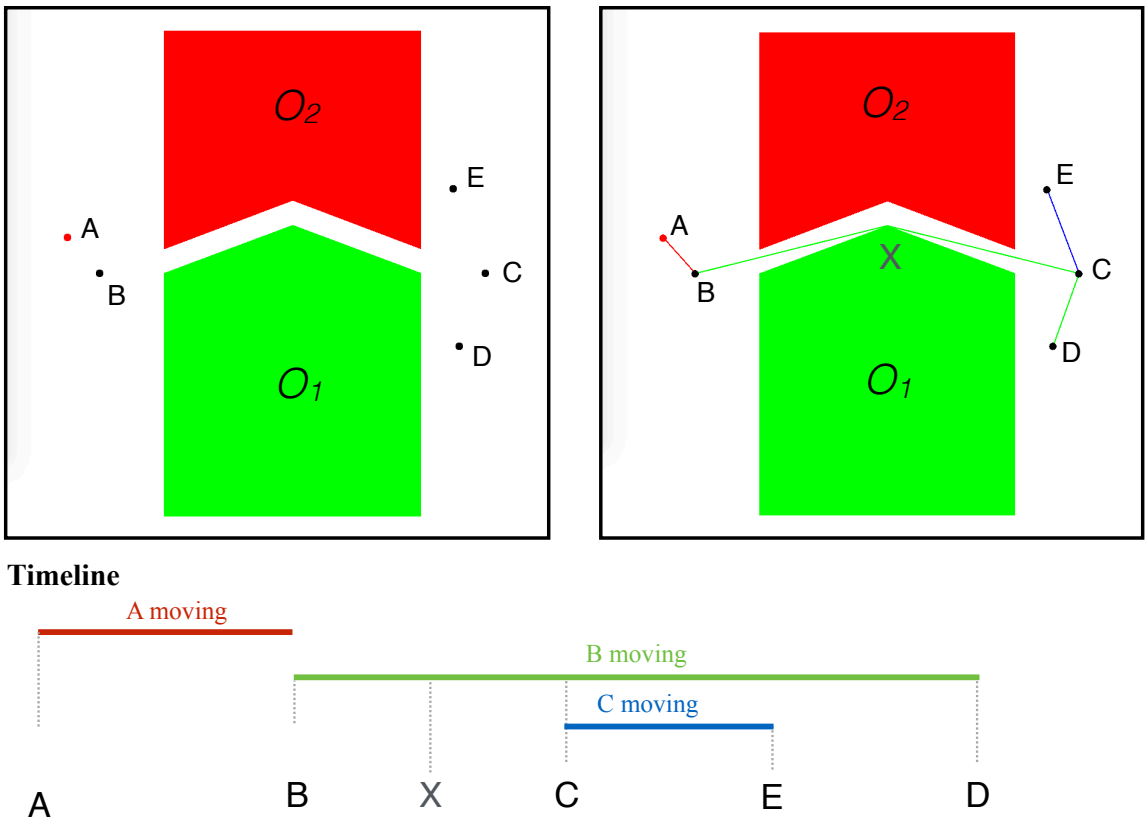

Figure 2: Moving and tagging with obstacles, starting from A: initial setup, paths and timeline.

2 Tasks and Grading

For this week, you will be working in groups of up to four members. Each group is assigned a **unique identifier** (a name of an animal species) and a **password**, which will be mailed to you at the beginning of the week. As customary, we advise you to keep your password secret, as it will be used for submission of the results by your team to the testing server.

There will be four tasks of different difficulty, and we encourage you to make the best of splitting the workload to solve them in parallel. The maximal grade for this scenario week is **100 points**. The table below outlines the distribution of the points between several tasks:

Task Name	**Maximal Score (points)**	**Details**
Solving 30 large instances of the MAT problem	60	Section 2.1
Visualisation of the Solutions	10	Section 2.2
Implementation Report	10	Section 2.3
The Competition	20	Section 2.4

The next subsections outline the details of each of the tasks.

2.1 Solving the Move-and-Tag problem

In this task, you will be given a text file `robots.mat` (available from the Moodle page of the course), containing definitions of **30 configurations with robot positions and obstacles**. The obstacles are simple polygons, with no holes or self-intersections. They also do not intersect with each other.

Each line of the problem file contains an problem instance number, followed by a colon (ignoring possible spaces between other lexical tokens), a list of robot positons, a an optional `#` separator followed by a list of polygons separated by a semicolon, if there are any obstacles.

Each robot is represented by a point on a plane, encoded as a pair of integer or *double-precision* floating-point coordinates, e.g., `(4.5, 3.534635257)` or `(5,0)`. A polygon is represented by a list of coordinates of its vertices (x, y), where x, y can be integers or double-precision floating-point numbers. The sequence of the vertices is arranged in a way that the interior of the polygon will stay *on the left*, when one "walks" from one vertex to the next one. The successor of the last vertex in the list is the first vertex.

For instance, the following text describes the two problems from Figures 1 and 2 in the defined format, numbered 1 and 2 correspondingly:

```
1:  (-1.5, 1.5), (-1,0), (5,0), (4.5, 3.5), (4.6, -3)
2:  (-1.5, 1.5), (-1,0), (5,0), (4.5, 3.5), (4.6, -3) # (0,1), (2,3), (4,1), (4,10), (0,10); (4,0), (2,2), (0,0), (0,-10), (4,-10)
```

The first line corresponds to the obstacle-free setup from Figure 1, with the five coordinates describing positions of the robots A–E, so the robot A with coordinates `(-1.5, 1.5)` is the first one in the list, so it should awake the rest. The second line describes the setup from Figure 2, with the same robots and two obstacles, represented by 5-vertex polygons.

Your goal for this task is to compute, for each MAT instance, a set of paths of a *subset* of robots, so by following those paths the corresponding robots would eventually awake all their peers without forgetting anyone and without intersecting the obstacles. You only need to give paths for robots that *do move* in your solution.

The solution for this task is a text file. You can implement your algorithm in any programming language of your preference and use any libraries you consider necessary. You do not have to (and should not) submit the code.

The file with the results should start with the first line containing the name of the team and the second line being its password. If those do not match, the file will not be accepted by the system. The remaining lines should contain the solutions in the format, described by the following grammar (white spaces are ignored and can be added arbitrarily):

```
<ProblemSolution> := <number>: <Solution>
<Solution> := <path> | <Solution> ";" <path>
<path> := <points>
<points> := <point> | <points> "," <point>
<point> := "(" <float> "," <float> ")"
```

A solution for each problem, along with its number, should be placed on a separate line. There is no specific order imposed on the sequence of the paths or solutions. Each solution line starts with a number of a problem, followed by a semicolon, followed by one or more paths (lists of points), separated by semicolons. Each path should be a list of points, described the movements of a specific robot. It should start from a a coordinate of some robot, and explicitly mention all other robots,

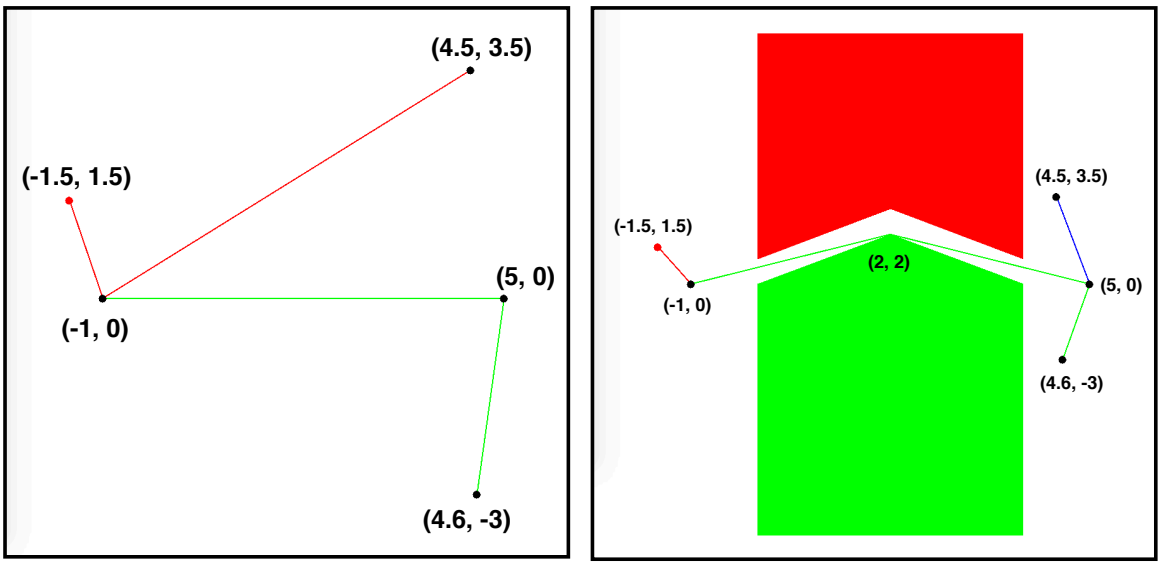

Figure 3: Problems from Figure 1 and Figure 2 with the corresponding coordinates.

which this one is going to "tag", as waypoints, as well as points where direction is changed in order to avoid obstacles. The ordering of points in a path is important, but the ordering of paths in a solution does not matter. In each solution, there must be exactly one path that starts with coordinates of the first robot.

For instance, solutions for the problems from Figures 1 and 2 submitted by the team `tiger` with a password `lt671vecrskq` might look as follows:

```
tiger
lt671vecrskq
1:  (-1.5, 1.5), (-1, 0), (4.5, 3.5); (-1, 0), (5, 0), (4.6, -3)
2:  (-1.5, 1.5), (-1, 0); (-1, 0), (2, 2), (5, 0), (4.6, -3); (5, 0), (4.5, 3.5)
```

That is, the solution for the first problem contains two paths: for the robot A (with coordinates `(-1.5, 1.5)`) and for the robot B (coordinates `(-1, 0)`), which will wake up all other robots. Similarly, the second line provides three paths outlining the routes for the robots A, B and C. The graphical representation of the solutions, with the corresponding coordinates, is shown in Figure 3.

Each solution will be assigned a ***score*** (rounded up to 0.001), corresponding to the actual time of the swarm awakening according ot the provided paths, computed as a length of the joint timeline (1 unit of distance = 1 unit of time), where several robots can be working in parallel. For instance, the score assigned to the first solution from the example above is `10.608`, whereas the score of the second one is `12.327`, because of the detours, required to go around obstacles.

The text file with the solution should be submitted in the form of the following page:

`http://scenario.cs.ucl.ac.uk`

WARNING! Make sure that the coordinates of the robots and polygon vertices in your solution are *exactly* those that are given in the problem file. For instance the coordinates `(1.0, 1.0)` and `(1.0, 1.00000000001)` are considered different points by the solution checker!

WARNING! Parts of the input are specified via *double-precision floating points*, which assumes working with ε-equality instead of equality.[1] Your solutions may contain double-precision floating-point numbers, as well. The server uses $\varepsilon = 0.000000001$, therefore all values with difference smaller than 0.000000001 will be considered *equal*. This value of ε is unsound for arbitrary floating-point computations, but should suffice for the solutions of the problems in this scenario.

A submitted file might not contain *all* solutions, so only presented ones will be graded by the system. Files with typesetting errors (i.e., no following the grammar outlined above) will not be accepted for grading. A solution for a specific problem instance **will not** be accepted if at least one of the following conditions holds:

1. Some of robots' paths intersect the obstacles (it is okay to touch the boundaries);
2. Not all of the robots in the swarm should are awakened upon executing all of the paths;
3. The are "cycles" in the awakening dependencies;
4. There are paths with duplicated starting point—one robot can only start one path.

Grading a full submission with solutions to all 30 instances typically takes about a minute, and might take significantly longer, depending on the current server load. Once grading is complete, the statistics for the solution will appear in the joint score table. Notice that only the **last** solution is taken into account, so all previous results for a team are superseded by the next submitted solution.

The server will stop accepting solutions at 14:00 GMT, 24 February 2017. Make sure to submit your best results by then.

The maximal grade you can get for this part is **60 points**: two points for each solved MAT instance.

2.2 Visualisation of the Solutions

You can provide an application for visualising the polygons and the results of your algorithms (it will also make testing of your solutions easier). The app does not have to feature sophisticated input components and can just take text files, similar to the ones describing problems and their solutions. Your implementation of the visualiser should feature the following elements:

1. drawing the obstacle shapes;
2. drawing initial robot positions;
3. drawing paths of different robots in a way that allows to identify which robot is moving, for instance, via colours or motion.

The applications should be demonstrated to TAs in a dedicated time slot between 14:00 and 17:00, 24 February 2017.
Book your team a slot on Moodle

The maximal grade you can get for this part is **10 points**.

[1] `https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/`

2.3 Implementation Report

None of the previous assignments required you to submit any working code. However, to get additional points you can submit a short report, describing your implementations of the procedures for solving the MAT problem. In particular, your report should outline the following components of your development:

1. Which language has been used to implement the algorithms and what external libraries were used? Which libraries were used to implement the visualisation?
2. Which geometric algorithms (including auxiliary ones) were implemented/invented for constructing the computing the robot paths?
3. What is the complexity and the observed run time for the implemented solutions?
4. How were the algorithms for solving MAT tested?
5. How were the input files processed, and output files produced?
6. How was the design/implementation workload split between the members of the team?
7. A link to the repository with the development (can be made public after the scenario week).

The reports should be submitted to the Moodle page.
by 17:00, 24 February 2017.

The maximal grade you can get for this part is **10 points**.

2.4 The Competition

Even though any valid solution for a corresponding MAT instance will be accepted, the teams should strive to deliver the best solutions. To stimulate the search for better MAT-solving algorithms, the per-team submissions will be ranked according to the scores assigned to solutions for specific instances, as described above (smaller score is better). The total team ranks computed out of the last submissions are displayed in a joint score table. Teams that submitted acceptable solutions for all the problems will be generally ranked higher than teams whose submissions are partial. In the case of equal results (modulo rounding up to 0.001), the same rank will be assigned to multiple teams. You can earn up to **20 points** for this part according to the following formula:

$$Reward(team) = 20 - \min(20, rank(team) - 1)$$

Therefore, the team(s) holding the first place will earn the maximum of 20 points, whereas the teams ranked 21 and more will not earn any points for the competition.

Good Luck!

3 Useful Resources

- Robot motion planning: `http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf`
- Freeze-tag problem
  `http://cs.smith.edu/~jorourke/TOPP/P35.html#Problem.35`

4 Frequently Asked Questions

Q: We have a problem with a task and/or do not understand the problem and the requirements for the solutions. How can we clarify this?

A: Ask a TA during the dedicated sessions or send an e-mail to `scenario@cs.ucl.ac.uk`. The first option is preferable, as your e-mail might be answered only in a few hours.

Q: What are the deadlines for the automatically-checked solutions?

A: The deadline for the MAT solutions and the Competition is **14:00 GMT** ($\pm\varepsilon$), **Feb 24, 2017**. After that moment the server `http://scenario.cs.ucl.ac.uk` will stop accepting solutions.

Q: We did not get an email about our submission. What should we do?

A: Grading solutions to 30 problems on an "empty" server takes about a minute. Under high loads, processing your solution might take up to half an hour. If the results do not appear in a score table by then and you do not get a notification email, please, contact the Scenario Week organisers (`scenario@cs.ucl.ac.uk`).

Q: Does the server keep the best solution we've ever submitted?

A: No, the server only takes the *last* submitted solution into the account, so make sure that your solution file contains solutions for as many MAT instances as possible (ideally, all of them).

Q: We got an email reporting some errors in our solution. However, the scoreboard also shows some parts as "N/A", even though no issues were reported for them. How so?

A: Your submission file might be missing solutions for specific instances. In the case of errors in the solution, only those are reported in an automatic email, without mentioning missing parts of the submitted file. Missing parts are only reported if the rest of the solution file is okay.

Q: Can we submit empty solutions for some problems (e.g., having lines like `20:  `)?

A: No, the whole submission file will be then rejected by the server input validator. To submit a partial solution, just omit these parts from the file.

Q: Some of our solutions did not pass the check with the report "*There is a cycle in the solution where no robot gets awakened*", even though we are sure that there is no cycle! What's wrong?

A: Make sure that the coordinates of robots in your pats (e.g., the very first robot) are exactly the same as given in the problem file. For instance the coordinates `(1.0, 1.0)` and `(1.0, 1.00000000001)` are considered different points by the server, hence it draws the conclusion that some of the robots are missing from the paths.