The 2013 Multi-objective Physical Travelling Salesman Problem Competition

Diego Perez, Edward Powley, Daniel Whitehouse, Spyridon Samothrakis, Simon Lucas, Peter Cowling

***Abstract*— Numerous competitions have emerged in recent years... and this is another one.**

I. Introduction

Research in games and competitions. Literature review.

[1], [2].

II. The Multi-objective Travelling Salesman Problem

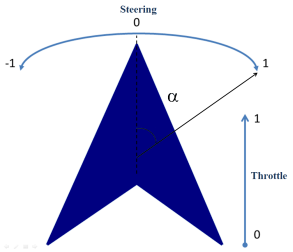
Description of the game: physics, constraints. Description of maps: dimensions, specifications.

The Multi-Objective Physical Travelling Salesman Problem (MO-PTSP) is a modification of the Physical Travelling Salesman Problem (PTSP), previously introduced by Perez et al. [1] for the WCCI 2012 PTSP Competition. The PTSP is a game where the player controls a ship with the goal of visiting 10 waypoints scattered around the maze in as little time as possible. MO-PTSP adds two more goals to the game: doing this spending as less fuel as possible and reducing the damage suffered by the ship. This section specifies the main components of the MO-PTSP game, such as the game physics, objectives, rules and maps.

A. Game Physics and the Real-Time Component

Both PTSP and MO-PTSP are games that share some characteristics. One of these similarities is the physics of the ship used by the player to navigate through the game. In both games, the agent controls a ship where two different inputs can be applied: `steering` and `throttle`. The first input can be set to *left*, *straight* and *right*, while the throttle input can be *on* or *off*. The combination of these inputs adds up to 6 different actions that can be provided at a given time. Figure 1 depicts the moves available for the ship.

Left and right rotations are performed using a single angle, set to $\pi/60$ radians, and ship speed is set to 0.025 pixels per time step, values determined by trial an error in order to provide a satisfactory game-play experience. The ship keeps its velocity from a game state to the next, making *inertia* a key aspect to take into account when governing the ship. However, the ship would reach a full stop eventually if no acceleration actions are provided, as there is a loss of speed due to *friction*. This friction is set to 0.99 (this is, only 1% of the speed is lost at each step), a value also determined empirically. Finally, there is also a speed gain for actions with throttle on, which is set to 0.025 pixels per game step.



Action ID	Steering	Throttle
0 (*Neutral* action)	*Straight* (0)	*Off* (0)
1	*Left* (-1)	*Off* (0)
2	*Right* (1)	*Off* (0)
3	*Straight* (0)	*On* (1)
4	*Left* (-1)	*On* (1)
5	*Right* (1)	*On* (1)

Fig. 1
Ship inputs and actions.

The location of the ship in the world can be uniquely determined by three vectors: *position*, that specifies the coordinates of the center of the ship in the maze; *direction*, a vector that indicates where the ship is pointing at (in other words, indicates the direction of a front vector); and *velocity*, that determines the movement of the ship, including its direction and speed. Note that direction and velocity do not need to be aligned. Given an action, this part of state of the ship is modified as shown in Algorithm 1.

Algorithm 1 Ship update function - no collisions.

```
function ShipUpdate(action)
    throttle ← action.GetThrottle()
    steering ← action.GetSteering()
    ship.direction.Rotates(steering × steerStep)
    if throttle == true then
        ship.velocity.Add(ship.direction × shipSpeed)
    ship.velocity.Multiply(frictionLoss)
    ship.position ← ship.position + ship.velocity
```

The ship can also hit obstacles while it moves within the level, so position and velocity need to be updated differently if the ship's circular bounding collision hits a obstacle in the maze. When this happens, speed is reduced (by a 75% factor) and the velocity vector is modified so the ship bounces off the wall with the appropriated angle. However, the MO-PTSP game introduces a new type of *elastic* obstacle. In the case when the ship hits this kind of obstacle, the reduction of the speed is minor (only 10%), so the player can use this type of walls to change direction of travel abruptly without losing too much speed. A third type of obstacle (known as

Diego Perez, Spyridon Samothrakis and Simon Lucas are with the School of Computer Science and Electronic Engineering of the University of Essex (email: {dperez, ssamot, sml}@essex.ac.uk).

Edward Powley, Daniel Whitehouse and Peter Cowling are with the Computer Science department of the University of York (email: {edward.powley, dw830, peter.cowling}@york.ac.uk).

This work was supported by EPSRC grant EP/H048588/1.

damaging obstacle), produces a more important decrease in the speed of the ship, reducing it by a factor of 90%.

Another similarity between PTSP and MO-PTSP is the real-time component of the game: the player (also referred here as the agent, or the controller) must supply an action within a limited budget time, set to 40 milliseconds. This limitation forces the controller to determine the next move quickly, rewarding those agents that are able to plan faster and explore the action search space in a more efficient manner.

While the PTSP was a bit more permissive, MO-PTSP imposes disqualifications in case this time limit is severely violated. In the original game, a neutral action was applied if the controller took more than 40 milliseconds to provide an action, but no other consequences were derived from this misbehaviour. The game was then susceptible to cheating, as an agent could employ as long as it needs to plan ahead while the only consequence would be a neutral action in a single game tick.

MO-PTSP tackles this situation by imposing a second time limit, set to 120 milliseconds so, in case it is violated, produces the end of the game by disqualifying the player. Although controllers could still potentially spend more than 40 milliseconds (with the same neutral move performed as a penalization), the risk of being disqualified and the limited gain that could be obtained by these extra milliseconds up to 120 discouraged participants to performed this trick.

B. A Multi-Objective Approach

The main difference with respect to the original game, the PTSP, can be found in the three different objectives that need to be minimized:

- Time: the player must collect all waypoints scattered around the maze in as less time steps (or game cycles) as possible.
- Fuel: the fuel consumed at the end of the game must be minimized.
- Damage: the ship should end the game with as little damage as possible.

It is very important to stress that all waypoints must be visited in order to consider a game as successfully finished. Otherwise, a very simple but plausible approach for the player can be not to move at all and hence obtain a good result for the other two objectives (no fuel spent and damage suffered). A time dependant game over condition is established when the ship does not visit a waypoint once a timer has run off. This timer, initially set to 800 time steps, is reduced by 1 at every step if no waypoint is visited, causing the end of the game if it gets to 0[1].

The ship starts with an initial fuel of 5000 units, and one unit is spent every time an action with the throttle input set to *on* is performed. The player adds, however, 50 more units to the ship's fuel tank every time a waypoint is visited. Also, four *fuel canisters* are also scattered around the maze, that provide 250 units of fuel. It is important to mention that it is not mandatory to pick these fuel canisters up in order to complete the game, being their collection up to the strategy of the player. In case the amount of fuel available in the ship gets to 0, the agent will not be able to use the throttle any more.

Regarding the third objective, the ship can be damaged by two different game play elements. First, lava lakes are present in the game levels, dealing 1 unit of damage for every game step the ship is flying over them. Secondly, the ship also gets damage when colliding with (non-elastic) obstacles, such normal and *damaging* obstacles. The former type of obstacles (normal walls, also present in the PTSP game) inflict 10 units of damage. The latter obstacles, created for this game, produce a more harming effect, dealing 30 damage units. If the player's damage gets up to 5000 units, the ship is destroyed and the game is over.

When the ship is damaged by a collision, it enters in an invulnerable state, where no more collision damage can be dealt to the agent (although lava still affects the ship) during 50 game steps. This avoids situations when too much damage is suffered by the ship if it is touching an obstacle right after colliding with it.

C. Game Maps

Each level for the MO-PTSP game follows a specific format, based on the one that Nathan Sturtevant defined and was employed in several games such as Warcraft, Starcraft or Baldur's Gate. This type of maps have been used by some researchers in the literature before [2]. The levels are stored in ASCII files, where each character determines the type of object or surface located at that pixel in the map grid. For MO-PTSP, these symbols are detailed next:

- Obstacles are represented by the characters 'T', 'D' and 'L', to be used to create normal, damaging and elastic obstacles respectively.
- Normal surfaces use the character '.', while lava lakes employ ':'.
- Waypoints and fuel canisters are indicated with 'C' and 'F' respectively.
- The player's ship is represented by the character 'S'.

Figure 2 presents an example of a MO-PTSP map, as it is drawn by the framework. Not visited waypoints are depicted as (filled) blue circles, while the already visited ones are shown as empty circles. Green ellipsis are used for the fuel canisters. Normal surface is brown, whereas lava is a red-dotted yellow surface. Elastic collisions are blue, normal collisions black and damaging collisions are drawn in red. The ship is drawn as a dark blue polygon, with a green triangle at the back denoting thrust being used. The trajectory followed by the ship is drawn with a black line.

III. The MO-PTSP Framework

A. Game flow

game flow

[1]PTSP also employed this timer, starting at 1000. Hence, MO-PTSP is slightly more difficult in this sense, as it requires faster controllers.

[2]http://movingai.com/benchmarks/dao/

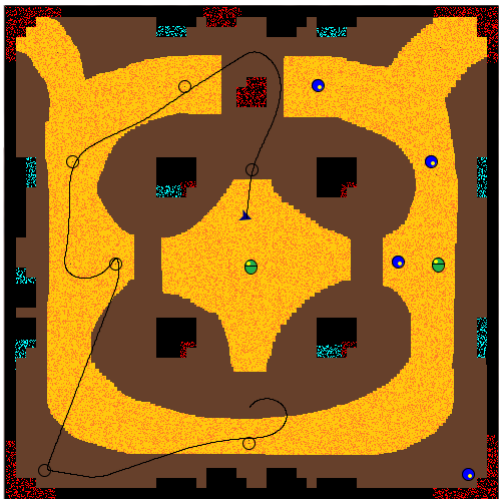


Fig. 2
SAMPLE MO-PTSP MAP.

B. Software

software organization

C. Controllers

Programming specification, API, language, pathfinding.

IV. THE MO-PTSP COMPETITION

A. Competition format

Infrastructure of the competition, rules

B. Submission and evaluation

Submission, Different phases, maps. Ranking in a map, ranking in the competition

C. Sample Controllers

Sample controllers: random, greedy, macro-action random search controller. Include some measurements.

V. RESULTS OF THE COMPETITION

Describe results in detail.

VI. THE WINNING ENTRY

YORK DESCRIPTION.

VII. CONCLUSIONS

We had lots of fun.

REFERENCES

[1] D. Perez, P. Rohlfshagen, and S. Lucas, "The Physical Travelling Salesman Problem: WCCI 2012 Competition," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2012.

[2] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4:1, pp. 1–43, 2012.