Completion on milestone

MAPF Project

Yonatan Shelach

Tom Seligman

Jessica Leibovich

Background

The purpose of our project is to solve the warehouse problem as described in our main report, using the Transformer model.

This report describes the completion of a milestone in our project:

As part of building the training data for our transformer model, we managed to use the LNS2 algorithm in a way that allowed us to create high-quality data for training that improves the results of our model.

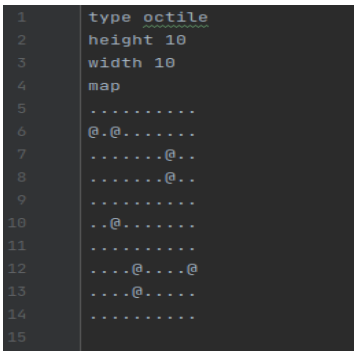
Description of the LNS2 algorithm

The LNS2 algorithm is an algorithm that solves the Multi-Agent Path Finding problem in a novel way. The way this algorithm works is by guessing a set of paths that contains collisions, and then repeatedly selecting a subset of colliding agents and replans their paths to reduce the number of collisions until the paths become collision-free.

The LNS2 algorithm receives as input:

- a file named $warehouse.map$ which describes the warehouse.

The file includes parameters such as height and width and each position in the warehouse is represented by a $'.'$ or $'@'$ were $'.'$ represents a passable position and $'@'$ represents an obstacle.
Here is an example of a warehouse representation:



- A file named $batch\_\{number\}.scen$ where {number} is the number of the routing requests batch.

We run the LNS2 multiple times with different routing requests batches. Each batch consists of $K + 1$ lines where K is the number of requests in a batch.

Each request is represented by 9 numbers:

1. Bucket number – designed to improve the performance of the LNS2 but we set all buckets to zero as this improvement is minuscule when considering the number of requests, we set in a batch.
2. Name of the warehouse map file.
3, 4. Dimensions of the warehouse.
5,6,7,8. The coordinates of the starting position and the target position for the agent.

Notice:

Unlike in our MAPF problem, the LNS2 algorithm receives routing requests that don't include a constraint on the arrival time.

The output of the LNS2 algorithm is a file called $batch\_\{number\}$ corresponding to the input file with the same number. This file includes a lot of statistical information produced by the algorithm, and a solution for the routing requests in the LNS2 format which represents each step in the path with one position number, following the formula:

$$position = (warehouse\ width) \times (x\ coordinate) + (y\ coordinate)$$

Using LNS2 to create our training data

The training data for our transformer model is consistent of batches of K requests where each request is described by five numbers:

x_{start}, y_{start} – the coordinates of a starting position for an agent where y_{start} is always at the beginning of the warehouse.

x_{end}, y_{end} – the coordinates of a target position for an agent where y_{end} is always at the end of the warehouse.

$arrival\_time$ – The time in which the agent is required to arrive from position (x_{start}, y_{start}) to position (x_{end}, y_{end}).

We wanted to make sure that the value we set for $arrival\_time$ is reasonable when considering the starting and target positions.

We achieved so, this by setting the starting and target positions first in a random way and then finding a solution for the request without the time constraint using the LNS2 algorithm.

Then, we used the lengths of the paths in the LNS2 solution as the arrival time for our routing requests. Those, ensure that we possess a new batch of routing requests, in our format (with arrival time constraint), that has a solution (the same as the LNS2 solution).

The flow of the code for creating the training data is:

1. Creating many batches of input files ($batch\_\{number\}.secen$) consisting of routing requests without time in the LNS2 format, produced with random values for: x_{start}, x_{end} and the required values by our model for: y_{start}, y_{end}.

2. Running the LNS2 algorithm on all our batch files. We used multi-threading which significantly reduced the running time for this stage.

3. Reading and converting the solution in the LNS2 output files into solutions in our format which represents each step in the path by its coordinates.

4. Validating the solutions from the previews step – including validation of the paths and that there are no collisions in the given solution.

5. Creating the routing requests batches in our format based on the valid solutions that ensure that our routing requests batches are solvable.

6. The final output file with the training data is a CSV file called: $training\_data.csv$

Steps 1 and 2 are done by the file $lns\_io.py$.

Step 3 is done by the file $solution\_validator.py$.

Steps 3,5 and 6 are done by the file $training\_io.py$.

The parameters of the warehouse, and for the creation of the training data are all set and programmable in the file: $config.py$.

We also created a file called debugging_utils.py which contains useful methods for debugging.

Challenges we encountered:

Unlike in our model, the LNS2 algorithm allows for turns at the end of the warehouse, which is why we add 1 to the arrival time produced by the described procedure.

Also, the LNS2 algorithm don't allow two requirements with the same target.

To combat that, we recommend doubling the target options from the desired amount and then using its LNS2 solution for calculating the arrival time.

That way, we can approximately simulate having multiple requests with the same target.