Flatland encoding

> Week 6

Progress

Encoding

- Last week, we had a bug last week that prevented it from scaling up
  - There was a problem with the transitions made in the Python with the boundaries
  - Now it can efficiently handle a 70 x 70 environment with 4 agents
- We have a maxTime() which at the moment is the same for all trains
- We have a line that stops the train at the goal, so that it doesn't force the action to continue going all the way until the max time
- Constraints are mainly the same
- A score which counts how long all agents take, everything totaled together

As a result, we could have many possible plans because trains could wait at any time.

Python notebook

- Compares all models until it finds the best - we can use a minimize statement in Clingo
- Visualizer works (though may be off by a time step, for some reason)

Currently

1. We have multiple trains
2. We abandon the long edges
3. We do not have collisions
4. We have the visualizer
5. On short edges, we can track where the trains are at any time point

Next steps for Week 6

- On the long edges, check for collisions between multiple trains.
- Can Flatland generate deadlines?
    - There should be a one-line implementation of this
- One problem is the wait (#minimize statement)
    - We might need this even if we already have an optimal path for a single agent
    - We can't say "just wait at the end"
    - Try to minimize the number of action steps or the number of waits (this may effectively be the same). Since there are no actions once a train reaches its goal, we should be able to achieve this by minimizing the number of actions.
    - We want to avoid the train waiting for absolutely no reason. The train will, however, likely need to wait at some point.

An explanation of the minimize statement can be found in the most recent version of the Potassco guide under section **3.1.13 Optimization**.

As an alternative way to express an optimization problem, there are optimization statements. A minimize statement of the form

$$\texttt{\#minimize}\ \{\ w_1@p_1,\boldsymbol{t}_1:\boldsymbol{L}_1,\ldots,w_n@p_n,\boldsymbol{t}_n:\boldsymbol{L}_n\ \}.$$

represents the following n weak constraints:

$$\texttt{:\sim}\ \boldsymbol{L}_1.\ [w_1@p_1,\boldsymbol{t}_1]\quad\ldots\quad\texttt{:\sim}\ \boldsymbol{L}_n.\ [w_n@p_n,\boldsymbol{t}_n]$$

Moreover, maximize statements can be viewed as minimize statements with inverse weights. Hence, a maximize statement of the form

$$\texttt{\#maximize}\ \{\ w_1@p_1,\boldsymbol{t}_1:\boldsymbol{L}_1,\ldots,w_n@p_n,\boldsymbol{t}_n:\boldsymbol{L}_n\ \}.$$

represents the following n weak constraints:

$$\texttt{:\sim}\ \boldsymbol{L}_1.\ [-w_1@p_1,\boldsymbol{t}_1]\quad\ldots\quad\texttt{:\sim}\ \boldsymbol{L}_n.\ [-w_n@p_n,\boldsymbol{t}_n]$$

As with weak constraints, the priorities '$@p_i$' are optional and default to zero.