TraceGen: Documentation

Bikramjit Banerjee
School of Computing
The University of Southern Mississippi
Hattiesburg, MS 39406-0001
Bikramjit.Banerjee@usm.edu

1 Introduction

This document describes the TraceGen codebase and its input language. TraceGen can be used to generate problem instances for plan recognition of one or more agents. In particular, it generates the activity traces of $n \geq 1$ agents for T steps of time, grouped into random and dynamic teams, and capable of interleaving activities from multiple plans.

2 Input Language

The input is a file that contains text encoding of a series of partially ordered plan graphs. A plan graph may look like the following:

```
START.AKT|HSC|UR
GOAL.STAR
STEP.0.(noop)
STEP.1.(stack s t)
STEP.2.(stack t a)
STEP.3.(stack a r)
STEP.4.(unstack a k)
STEP.5.(pick-up t)
STEP.6.(unstack s c)
STEP.7.(unstack h s)
STEP.8.(unstack k t)
STEP.9.(unstack u r)
STEP.10.(put-down h)
STEP.11.(put-down u)
STEP.12.(put-down k)
ORDER.2<1.3<2.4<3.4<8.5<2.6<1.7<6.7<10.8<5.8<12.9<3.9<11
ROLE.1=6.2=5.3=4.7=10.8=12.9=11
INTERLEAV_PTS.1.2.3.10.11.12
```

This is a blocks-words plan, where a bunch of blocks (each labeled by an alphabetic character) are given in some initial configuration, and the goal is to spell out the goal-word, not caring about the blocks unused in this word. Figure 1 shows the corresponding picture for clarity.

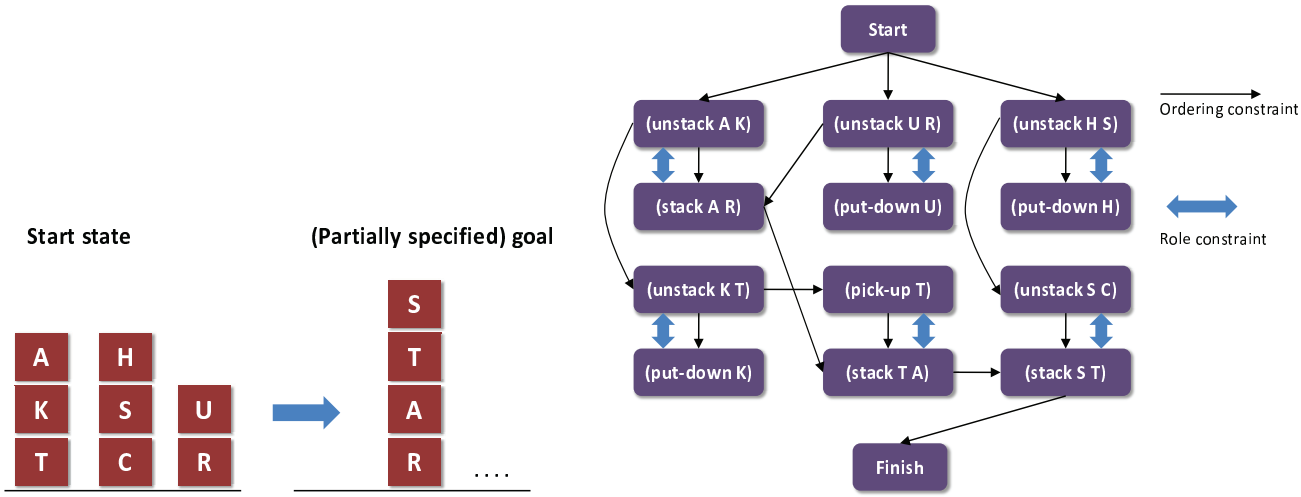

Figure 1: Multi-Agent Blocks Word. (a) shows the start and goal state of the problem. (b) shows the corresponding plan graph whose text description is given above.

The first two lines, `START.x` and `GOAL.y`, describe the start state and the goal of the plan. `x` and `y` can be any strings void of the period, and are used to identify plans in the explanation file produced as output. This assumes that there can be only one plan graph for each (start, goal) pair. Relaxing this assumption is a future work.

The 3rd line *must* be `STEP.0.(noop)`. "(noop)" (short for "no operation") is the catch-all for idling and extraneous actiions.

From the fourth line onward the various steps of the plan are enumerated in any order. These are the vertices of the plan graph. Their partial ordering is specified in the line starting with `ORDER`. Thus each component `x<y` in this line is a directed edge (ordering constraint) from vertex `x` to `y` in this graph. The line starting with `ROLE` specifies the role constraints, i.e., which plan steps must be executed by the same agent. Thus each component `x=y` in this line is a bidirectional edge (role constraint) between vertices `x` and `y`. Since the lines starting with `ORDER` and `ROLE` represent constraints, you can leave the rest of these lines empty to indicate the absence of constraints.

The line beginning with `INTERLEAV_PTS` is *not* about constraints. The `INTERLEAV_PTS` line specifies the steps after which an agent is allowed to switch plans for interleaved execution. E.g., in the above example, an agent could only switch if it is not holding a block, i.e., right after steps 1, 2, 3, 10, 11, or 12. But following step 4, for instance, it is holding block 'A', hence it cannot switch to a different plan until it has executed step 3. If the rest of this line is empty then TraceGen will assume that interleaving is impossible.

3 How To Run It And What Does It Do?

The file "sample_input" is included as an example of the input file of plan graphs. First run

```
python library_gen.py <input_file> [k]
```

to produce a file called "allPlans.dat". This step performs a limited combinatorial search to produce a small number ($3x$) of concrete plans that can be executed by x agent(s), for each $x \in [1, N]$. N is the maximum number of agents in a team that can execute a given plan graph (no sleeping partners allowed). This is calculated as the number of connected components in the plan graph with *only* role edges. If there are no role edges in your graph, then N will be equal to the number of steps in the plan, which may be too large for combinatorial search. In this case, you may limit the maximum team size with the optional argument `k` above. Although this program may take a while to run, you only need to run this once for each library of plan graphs (you may want to save the previous allPlans.dat generated from a different library under a different name so it is not overwritten).

Once the concrete plans have been produced in 'allPlans.dat', they can be used to generate the traces and the explanations. To do this, run

```
python trace_gen.py <input_file> <n> <T> <exp_file> <tr_file> <s> <p>
```

where

- `n` = number of agents (≥ 1)
- `T` = number of steps in the trace (≥ 1)
- `exp_file` = output filename where the explanation will be written
- `tr_file` = output filename where the trace will be written
- `s` = seed of the random number generator for reproducibility
- `p` = probability with which a team will interleave plan executions

The last parameter, `p`, is ignored (assumed to be 0) if the rest of the `INTERLEAV_PTS` line in `<input_file>` is empty. This step partitions the agent set randomly, then for each team it assigns a (start, goal) pair and selects a concrete plan for this (start, goal) and the team size. While the agents of a team execute the plan (and populate the trace), with probability `p` a branch point is effected if all agents in the team are at an interleav_pt. The current plan is suspended and either a new plan is started or a previously suspended plan is resumed. This continues until the team has either completed all plans (active and all suspended plans) or has reached `T`, the trace horizon. The agents are then free to join other teams. The relevant details of this process are recorded in the explanation file, `exp_file`.

The trace file, `tr_file`, is a comma separated matrix where rows correspond to time and columns correspond to agents. It is best viewed as a spreadsheet.

3.1 If working with a single agent ...

... then you should run

```
python library_gen.py <input_file> 1
python trace_gen.py <input_file> <n> <T> <exp_file> <tr_file> <s> <p>
```

4 Future Work

One main future task is to produce the plan graphs file automatically, given PDDL descriptions of a problem domain and a series of start-goal descriptions. Although much of this is automated, we still perform a crucial step of this process manually, hence we have not included that part of the code.

Another future task is to allow *unsynchronized interleaving*. The current version allows only synchronized interleaving, i.e. all agents in a team must switch to the same plan at the same time. We will relax this assumption.

The current code also does not accommodate negative constraints (e.g., negative role constraint (x, y) would mean steps x and y must *not* be done by the same agents. It also does not accommodate *concurrency constraints*, which would say which pairs of steps should be executed concurrently.

5 Questions, Comments

Please drop me an email to `Bikramjit.Banerjee@usm.edu` if you have any question or comment. Also, if you use the codebase please let me know.