Plan Library Generator

Giovani P. Farias

Pontifical Catholic University of Rio Grande do Sul – PUCRS
giovani.farias@acad.pucrs.br

February 24, 2017

PLGenerator (Plan Library Generator) is a package to automatically generate arbitrarily complex plan libraries [1, 2]. Such plan library generation can be directed through a number of parameters, creating a set of top-level plan trees, in a XML file format, which can be used as input to plan recognition programs. A plan library is a knowledge base that codifies in some way the agent's beliefs concerning how the agent can reach each particular goal in the domain. Typically, a plan library has a single dummy root node, where its children are top-level plans (goals) and all other nodes are simply plan steps. In the library, sequential edges specify the expected temporal order of a plan execution sequence whereas decomposition edges link sub-steps which are an elaboration (expansion) of the given plan step. The library has no hierarchical cycles, however, plans may have a sequential self-cycle, allowing a plan step to be executed during multiple subsequent timestamps, besides, each plan step has an associated set of observation features status of the agent and its actions.

1 How to run the PLGenerator.jar

The `PLGenerator` is packaged as a jar file able to produce a plan library based on various parameters, and is invoked from the command line as

```
java -jar PLGenerator.jar [OPTION]...
```

The parameters used to generate a random plan library are as follows:

- `(-g) number of top-level plans (goals)` represents the number of children for the root node, i.e., the number of different independent top-level plans (goals) in a plan library, presented as gray circles in Figure 1. Constraints: `(-g` $\geq$ `1)` — default `(-g 3)`
- `(-d) depth` of plan trees. In Figure 1 the number of nodes in a complete path from a top-level node to a leaf node is equal to `(-d 3)`. Constraints: `(-d` $\geq$ `1)` — default `(-d 4)`
- `(-b) minimum number of branches` that all nodes, other than root and leaf nodes, must have. Figure 1, all nodes have at least one branch `(-b 1)`. Constraints: `(1` $\leq$ `-b` $\leq$ `-B)` — default `(-b 1)`

- `(-B) maximum number of branches` that all nodes may have. The number of branches from a node is randomly chosen from interval `[-b; -B]` whenever a new node is created. In Figure 1 node `ps2.1` has `(-B 3)`.
  Constraints: `(-b ≤ -B)` — default `(-B 3)`

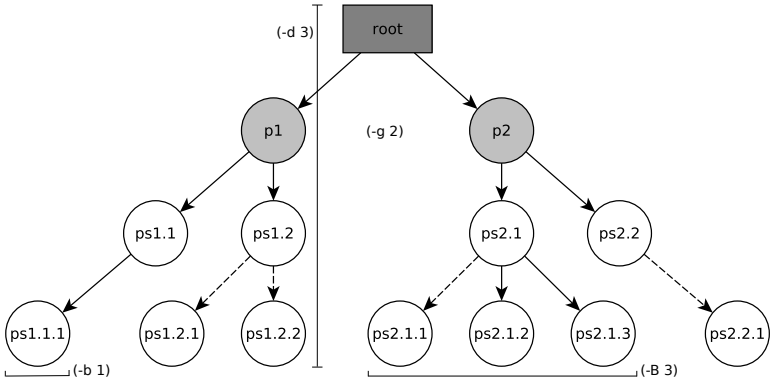

Figure 1: Ex. plan library with `(-g 2 -d 3 -b 1 -B 3)`.

- `(-f) features per node` represents the number of features associated with each plan node. Figure 2, each node must have at least one distinct feature `(-f 1)` to enable the recognition process. When a feature has no influence to recognize a specific plan step it receives a hash sign `‘‘#’’`.
  Constraints: `(-f ≥ 1)` — default `(-f 1)`

- `(-c) feature status (conditions)` represents the number of values that may be associated with the features. In Figure 2, considering `(-c 3)` means that all features are multivalued getting the value zero, one or two.
  Constraints: `(-c ≥ 1)` — default `(-c 2)`

- `(-F) number of features` available to be associated with a given plan node. In Figure 2, three features (f_1, f_2 and f_3) may be associated with plan nodes representing top-level plans or plan steps.
  Constraints: `(-F ≥ (-d)*(-f))` — default `(-F 10)`

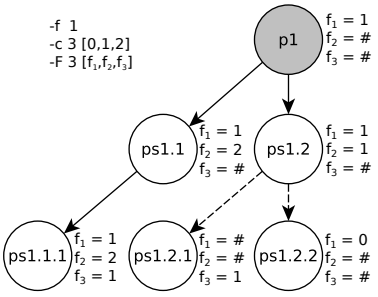

Figure 2: Part of a plan library with features valued `(-f 1 -c 3 -F 3)`.

- `(-s) sequential edges` determines the probability of a branch to be created as sequential (rather than decomposition). Thus, `(-s 0)` means that all branches will be decomposition type (Figure 3 (a)), whereas `(-s 1)` means that all branches will be sequential type (Figure 3 (c)).
  Constraints: `(0 ≤ -s ≤ 1)` — default `(-s 0.8)`

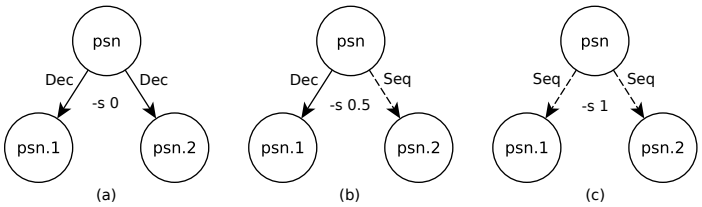

Figure 3: Examples of branch creation: solid lines are decomposition links (Dec) and dashed lines are sequential links (Seq).

- `(-D) duplication` represents the percentage of top plans (goals) duplicated in order to generate ambiguous paths. The duplicated plan is not exactly equal to the plan from which it was generated, because its leaf nodes are created different to establish some distinction between them (Figure 4). Example, `(-D 0.2)` means that 20% of top plans are a duplicate of another, thus approximately 40% of top plans are not unique, having some difference only in the leaf nodes.
  Constraints: `(0 ≤ -D ≤ 1)` — default `(-D 0)`

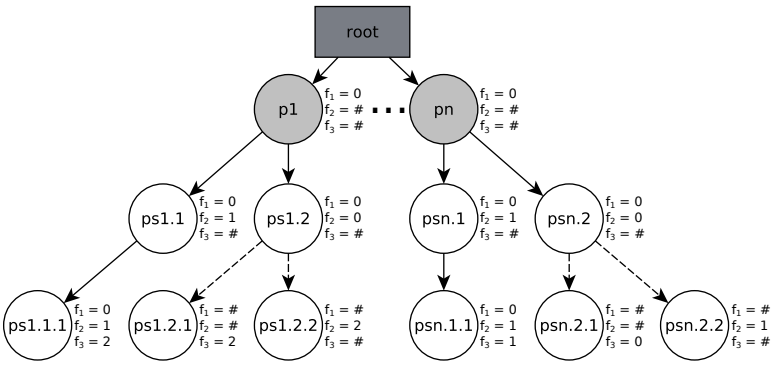

Figure 4: Example of duplication in a plan library where the top plan `pn` is a duplication of `p1` differing only in the leaf nodes.

- `(-t) minimum number of times` that an observation of a plan step must be repeated. Constraints: `(-t ≥ 1)` — default `(-t 1)`
- `(-T) maximum number of times` that an observation of a plan step may be repeated. Constraints: `(-T ≥ -t)` — default `(-T 1)`

- (-p) output file path to save the XML plan library.
  Constraints: (-p <String>) — default (-p plan_library.xml)
- (-A) Allows to set ALL parameters at one time in the following order:

  -g -d -b -B -f -c -F -s -D -t -T -p

  Example:
  java -jar PLGenerator.jar -A 3 4 1 3 1 2 10 0.8 0 1 1 pl.xml
- (-h) display the help content.

References

[1] Giovani Farias et al. "Automatic Generation of Plan Libraries for Plan Recognition Performance Evaluation". In: *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2015, Singapore, December 6-9, 2015 - Volume II*. 2015, pp. 129–132. DOI: 10.1109/WI-IAT.2015.55. URL: http://dx.doi.org/10.1109/WI-IAT.2015.55.

[2] Giovani Farias et al. "Evaluating the SBR Algorithm using Automatically Generated Plan Libraries". In: *5th Brazilian Conference on Intelligent System (BRACIS-16)*. Recife, Pernambuco, Brazil, 2016.