Short Introduction

1 Requirements

- GNU/Linux (Ubuntu 18.04 is recommended)
- Python 3.6 or newer
- CMake 2.8 or newer
- Modern C++ compiler that supports C++14
- Extra C++ Libraries (Optional). These libraries are placed in directory `third-party/`.
  - C++ Boost Library 1.65 or newer
  - pybind11
  - Google sparsehash-c11

Run following command in project root to install python packages

```
virtualenv --python $(which python3) venv
source venv/bin/activate

pip install -r requirements.txt
```

In this version of the hrg parser, **it uses a simple weighted model (similar to PCFG) to find the derivation in the forest with highest score.** The neural version of the decoder part will be released soon.

2 Induce Grammars

Run following command in project root to induce grammars from the sample data.

```
rules/extract_rules
```

Two grammars will be generated in directory `output/`.
All grammars are of the form `output/<name>.<date>.<graph>.<type>`.

- `name` is the name of the grammar (grammars named 'std' are used in our paper)
- `type` can be 'construction' or 'lexicalized'

`<grammar-dir>/train.mapping.txt*` are grammar files. `<grammar-dir>/train.graphs.txt` are eds graphs in the format which can be read by our C++ parser.
We also provide the grammars induced from the whole DeepBank1.1 training set.

- `data/construction.shrg.anonym` is constructional grammars.
- `data/lexicalized.shrg.anonym` is lexicalized grammars.

3 Build HRG Parser

Run following command in project root to build HRG parser:

```
bash build_parser
# Or run following script to build parser with python interface.
# <Python.h> is required in the host system.
bash build_parser_with_python # optional
```

Two files will be generated in directory `build`.

- `build/shrg_parser`: the main entry of parser
- `build/pyshrg.*.so`: the compiled python library file. (optional)

The python library file provides many APIs for manipulating the derivation forest.
See `parser/src/python/interface.cpp` for more information.

4 Run Examples

Available parser types:

- `tree_v1/best`, `tree_v1/naive`, `tree_v1/terminal_first`
- `tree_index_v1/best`, `tree_index_v1/naive`, `tree_index_v1/terminal_first`
- `tree_v2/best`, `tree_v2/naive`, `tree_v2/terminal_first`
- `tree_index_v2/best`, `tree_index_v2/naive`, `tree_index_v2/terminal_first`
- `linear`.

`v1` means to keep duplicated active items during parsing.
`v2` means to reject duplicated active items during parsing. But the parser will expand all active items in the derivation forest after parsing.
`index` means to use our indexing method.
`best` means to use tree decomposition with minimum width.
`naive` means to use naive tree decomposition.
`terminal_first` means to use a path decomposition described in our paper.
Run following command in project root:

```
# shrg_parser <paser_type> <grammar_path> <graph_path>
build/shrg_parser \
    tree_index_v2/best \
    data/construction.shrg.anonym \
    data/wsj.cpp.graphs
```

If the python interface is compiled, we can use following command to run parser on standard EDS format. (Remeber to copy `build/pyshrg.*.so` to `rules/`)

```
# Two new files `wsj.trees` and `wsj.output` will be generated
python3 rules/shrg_parser.py \
    data/construction.shrg.anonym \
    -P tree_index_v2/best \
    -i data/wsj.graphs \
```

```
    -C data/grammar.config \
    -o wsj
```

5 Note

Our implementation uses following two assumptions

1. **The input graph contains at most 256 edges**.
   We use an explicit set of edges to represent subgraph instead of the boundary representation. More precisely, we use `std::bitset<256>` to reprenset a subgraph, which is efficient enough for graphs in DeepBank1.1.

   Note that different representations of subgraphs only affect the parsing time. The number of successful/total item integrations should not be changed.

2. **Any RHS of the productions of the input grammar contains at most 16 nodes.**
   As a result, the number of external nodes of any partially recognized subgraphs is at most 16 during parsing. We use `std::array<std::uint8_t, 16>` to reprenset the mappings between boundary nodes.