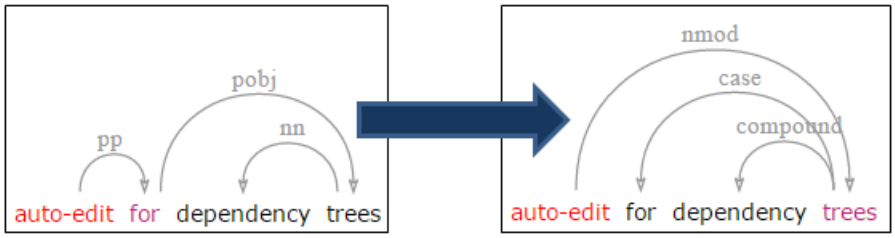

DepEdit
User Guide – Version 1.6.3

title:	DepEdit User Guide
software version:	1.6.3
guide version:	1.0.0
date:	19 October 2016

author:	Amir Zeldes
e-mail:	amir.zeldes@georgetown.edu
homepage:	http://corpling.uis.georgetown.edu/depedit/

Introduction

DepEdit is a simple configurable tool for manipulating dependency trees, written for Python 2.X and 3.X. To run it you need Python, a configuration file describing the manipulations (see Configuration), and an input file in the CoNLL 10 column dependency format (see Format; there is also limited support for other configurations). Basic usage is:

```
Python depedit.py -c config_file.ini INPUT.conll10 > OUTPUT.conll10
```

DepEdit can also be imported as a module into other projects to preprocess dependency trees (see Importing as a module).

Format

DepEdit uses the 10 column[1] version of the CoNLL dependency format, with some specific names given to fields in the configuration files. There have been different ways of using the CoNLL format in different projects, e.g. whether or not each second column in a pair is used for gold vs. prediction or some other purpose, fine/coarse grained tags (in MALT), lemmas in the third column, non-projective vs. projective dependencies, etc. DepEdit uses the following column names/mappings, though you may use these column names to run the script even if your own use differs. The following two examples show a minimal and elaborate use of the columns.

Basic input:

The 10 columns are tab-separated, sentences are separated by a blank line. Columns are (bold italics indicate column can be manipulated directly, see Configuration):

1. ***num*** - Token ID within sentence
2. ***text*** - Token text
3. blank (underscore)
4. ***pos*** - Part of speech
5. blank (underscore)
6. blank (underscore)
7. ID of head token
8. ***func*** – dependency function
9.-10. – reserved for alternate trees with multiple parentage (see below)

1	Wikinews	_	NP	_	_	2	nsubj	_	_
2	interviews	_	VVZ	_	_	0	root	_	_
3	President	_	NN	_	_	2	dobj	_	_
4	of	_	IN	_	_	3	prep	_	_
5	the	_	DT	_	_	7	det	_	_

[1] Starting in version 1.5.1, DepEdit also accepts an 8 column format without the last columns for multiple parentage/function. The last two columns can simply be omitted in such cases.

6	International	_	NP	_	_	7	amod	_	_
7	Brotherhood	_	NP	_	_	4	pobj	_	_
8	of	_	IN	_	_	7	prep	_	_
9	Magicians	_	NPS	_	_	8	pobj	_	_
1	Wednesday	_	NP	_	_	0	root	_	_
2	,	_	,	_	_	0	punct	_	_
3	October	_	NP	_	_	4	nn	_	_
4	9	_	CD	_	_	1	appos	_	_
5	,	_	,	_	_	0	punct	_	_
6	2013	_	CD	_	_	3	tmod	_	_

More elaborate input

This German example also encodes lemmas and morphology in the format, using column 2 (***lemma***) and column 6 (***morph***). A second POS column can also be used in column 5 (***cpos***), which is sometimes used for coarse grained or some alternative POS tag set.

1	Die	die	ART	ART	ART.Def.Nom.Pl.*	2	DET	_	_
2	Jugendlichen	Jugendliche	NN	NN	N.Reg.Nom.Pl.*	5	SUBJ	_	_
3	in	in	APPR	APPR	APPR	2	PP	_	_
4	Zossen	Zossen	NN	NN	N.Name.Dat.Sg.Neut	3	PN	_	_
5	wollen	wollen	VMFIN	VMFIN	VFIN.Mod.3.Pl.Pres.Ind	0	S	_	_
6	ein	eine	ART	ART	ART.Indef.Acc.Sg.Neut	7	DET	_	_
7	Musikcafé	Café	NN	NN	N.Reg.Acc.Sg.Neut	5	OBJA	_	_
8	.	.	$.	$.	SYM.Pun.Sent	0	ROOT	_	_
1	Das	die	PDS	PDS	PRO.Dem.Subst.Acc.Sg.Neut	2	OBJA	_	_
2	forderten	fordern	VVFIN	VVFIN	VFIN.Full.3.Pl.Past.Ind	0	S	_	_
3	sie	sie	PPER	PPER	PRO.Pers.Subst.3.Nom.Pl.*	2	SUBJ	_	_
4	bei	bei	APPR	APPR	APPR	2	PP	_	_
5	der	die	ART	ART	ART.Def.Dat.Sg.Fem	8	DET	_	_
6	ersten	erst	ADJA	ADJA	ADJA.Pos.Dat.Sg.Fem	8	ATTR	_	_
7	Zossener	Zossener	NN	NN	ADJA.Pos.*.*.*	8	ATTR	_	_
8	Runde	Runde	NN	NN	N.Reg.Dat.Sg.Fem	4	PN	_	_
9	am	an	APPRART	APPRART	APPRART.Dat.Sg.Masc	2	PP	_	_
10	Dienstagabend	Abend	NN	NN	N.Reg.Dat.Sg.Masc	9	PN	_	_
11	.	.	$.	$.	SYM.Pun.Sent	0	ROOT	_	_

Biplanar trees

Some data sources make use of the last two columns to draw secondary edges, such as external subjects for infinitives (xsubj in Stanford Typed Dependencies) or other relations. In such cases,

column 9 gives the secondary head (***head2***) and column 10 gives the function for that edge (***func2***). DepEdit exposes these columns partly by using the head2 and func2 fields, but does not directly model the second tree structure. As a result, you may use head2 and func2 in conditions and actions, but not in relations (see below).

Configuration

The manipulations carried out by the script are defined in a configuration file. This is a simple text file with one instruction per line and optional blank lines and comments (beginning with ';' or '#'). Each instruction contains 3 columns, as in the following example:

config_file.ini

```
;Connect nouns to a preceding article or possessive pronoun with the 'det' function
pos=/DT|PRP\$/;pos=/NNS?/      #1.#2           #2>#1;#1:func=det

;Change to-infinitive from aux to mark
text=/^[Tt]o$/&func=/aux/      none            #1:func=mark
```

The first column describes the tokens to be matched using regular expressions.

- Constraints are given as regular expressions over the fields:
  - ***num*** (column 1 of CoNLL10)
  - ***text*** (column 2)
  - ***lemma*** (column 3)
  - ***pos*** (column 4)
  - ***cpos*** (column 5)
  - ***morph*** (column 6)
  - ***func*** (dependency function, column 8)
- Multiple tokens are separated by ';'
- You can specify multiple criteria using '&', as in the second rule
- You may specify negative criteria using !=, e.g. lemma!=/able/
- You can use capturing groups in parentheses, which will be referenceable in the actions (third) column

The middle column defines relationships between tokens. It refers to each token in the definition by number (#1, #2...) and specifies:

- Adjacency (.): #1.#2 means the first token in column 1 is followed by the second

- Distance (.n or .n,m): #1.4#2 means 4 tokens distance, and #1.1,4#2 means a distance of 1-4
- Parentage (>): #1>#2 means the first token in column 1 is the head of the second token (note: this only applies to the main tree in biplanar input; head2 information is **not** used to establish parentage)
- If the instruction refers to only one token, as in the second example, the middle column says 'none'.

The third column specifies what to do if a rule matches:

- Change a property of token:
  - ***text***
  - ***pos*** or ***cpos***
  - ***func*** or ***func2*** - dependency functions
  - ***morph*** – the morphological analysis
  - ***lemma***
  - ***head*** or ***head2***[2]
- Make some token in the definitions the head of another
- You can refer back to values in capturing groups from the first column by using the number of that group, e.g. $1:
  - text=/(.*)/&pos=/IN/ … #1:func=prep_$1
- You can also convert the contents of $1, $2 etc. to lower or upper case by using $1L (the contents of $1, in lower case), or $1U (for upper case)

Importing DepEdit

Starting in version 1.5.0 you can import depedit as an object into other projects using the DepEdit class, which expects a configuration file handle. You can then use run_depedit() on some input file handles without loading the configuration multiple times. Starting in version 1.6.0, the module is compatible with both Python 2.X and 3.X, and is available via PyPI.

To install the module via pip:

```
> pip install depedit
```

[2] Changing head and head2 to a literal number is supported, and is mainly useful for setting them to 0 (root). If you want to rewire the primary head of some token to be another token, use a dominance directive instead (#1>#2).

```
from depedit import DepEdit

config_file  = open("path/to/config.ini")
deped = DepEdit(config_file)
docs = ["path/to/infile1.conll10", "path/to/infile2.conll10"]
for doc in docs:
    infile = open(doc)
    result = deped.run_depedit(doc)
```

Alternatively, you can also create a configuration inside your module, without reading it from a text file. There are several ways of doing this, which all achieve the same result:

```
from depedit import DepEdit
d = DepEdit()

##############################
# Ways to add transformations:
##############################
# From a single string per instruction
d.add_transformation("pos=/V/\tnone\t#1:func=x")
# From args
d.add_transformation("pos=/V/\tnone\t#1:func=z","pos=/V/\tnone\t#1:func=y")
# From a list
d.add_transformation(["pos=/V/\tnone\t#1:func=a","pos=/V/\tnone\t#1:func=b"])
# From a dictionary
d.add_transformation({"node":"pos=/V/","rels": "none", "actions":"#1:pos=a"})
```