PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 100 OCTOBER 2013 41–50

DIMwid — Decoder Inspection for Moses (using Widgets)

Robin Kurtz, Nina Seemann, Fabienne Braune, Andreas Maletti

University of Stuttgart, Institute for Natural Language Processing
Pfaffenwaldring 5b, D-70569 Stuttgart, Germany

Abstract
The development of accurate machine translation systems requires detailed analyses of the recurring translation mistakes. However, the manual inspection of the decoder log files is a daunting task because of their sheer size and their uncomfortable format, in which the relevant data is widely spread. For all major platforms, DIMwid offers a graphical user interface that allows the quick inspection of the decoder stacks or chart cells for a given span in a uniform way. Currently, DIMwid can process the decoder log files of the phrase-based stack decoder and the syntax-based chart decoder inside the Moses framework.

1. Introduction

Statistical machine translation is the research area that concerns itself with the development of automatic translation systems for natural language text using statistical processes. The last decade saw significant progress in the translation quality due to improved models and the availability of huge parallel text corpora. These corpora are used to automatically obtain translation rules, which are then weighted according to their usefulness. In this way, the translation model is obtained. In the decoding step, this model is applied to an input sentence to produce a translation of it. Modern statistical machine translation systems, such as Google Translate, are widely used nowadays and offer reasonable access to foreign languages. Naturally, the translations produced by those automatic systems are not perfect yet, but the gist can often be understood.

 Corresponding author: kurtzrn@ims.uni-stuttgart.de
Cite as: Robin Kurtz, Nina Seemann, Fabienne Braune, Andreas Maletti. DIMwid — Decoder Inspection for Moses (using Widgets). The Prague Bulletin of Mathematical Linguistics No. 100, 2013, pp. 41–50.
doi: 10.2478/pralin-2013-0010.

Frameworks such as Moses (Koehn et al., 2007) allow the fast and simple development of state-of-the-art statistical machine translation systems. The natural first step towards improving such a system consists of a detailed analysis of the recurring errors occurring in the baseline system. Once the problematic translations are identified, we would like to investigate how the translation was obtained from the rules. Besides the rules that were used in the translation we would also like to identify the competing rules and check whether a more suitable translation is possible in principle. Finally, we need to find out why the problematic translation was preferred over better translations.

These analysis steps are fully supported by the Moses framework, but require a manual inspection of the trace log of the decoder. The trace contains all the relevant information in plain text and can easily be used by experienced Moses developers, who know what to look out for. For novices the trace is not accessible at all because of its cryptic format and its sheer size. Our open-source tool *DIMwid* addresses this problem by providing a graphical user interface that displays the trace in a more user-friendly manner. A chart displays all translation items grouped according to the source span that they cover. *DIMwid* can display all standard traces of the Moses decoders in this manner and we also added a new trace allowing us to better identify used rule in the syntax-based chart decoder.

1.1. Design Choices

DIMwid should allow inexperienced Moses users to identify problematic translations, inspect the decoder trace to find the problematic rules and even find the competing rules that would potentially enable a better translation. To this end, *DIMwid* displays the trace output, which consists of the decoder stack items or chart items, in a uniform chart-based format. Each item is associated to the source span that it covers. This deviates from the grouping used in the standard stack decoder, but makes the analysis simpler. In fact, our goal was to make the tool simple enough to be useful for instructors in class. Naturally, each item typically comes with a variety of additional information (such as partial scores) and we display this information in the detailed view. Using this information and knowledge about the general decoding process, we can reconstruct how the decoder processed the sentence and which alternatives were considered. To streamline the inspection process, *DIMwid* can load the trace of multiple input sentences and allows opening several detailed views of items, which allow an easy comparison.

Besides the core functionality, we aimed to make the tool open-source and available on all major operating systems, which we achieved using the graphical framework Qt and the programming language Python, which is one of the most commonly used programming languages. In addition, Python source-code is easily readable and thus a popular choice for open-source projects. Finally, Python also supports our last goal, which was to build the architecture such that extensions and adjustments can

Language/Framework	Minimal Version		
	Linux	MacOS	Windows
Python	2.7.3	2.7.3	2.7.5
Qt	4.8.4	4.8.2	4.0
PyQt	3.18.1	4.9.4	4.10.1

Table 1. List of required packages together with their minimal tested versions.

easily be made to support future decoders and to satisfy the specific analysis requirements of users.

1.2. Related Work

The statistical machine translation framework Joshua (Li et al., 2009) already offers a graphical tool (Weese and Callison-Burch, 2009) for analyzing the translations. Joshua uses a syntax-based translation model and the visualization shows the hypergraph of the n-best candidate translations obtained during decoding. Moses is often used for phrase-based machine translation and we decided to use a CYK-parsing like chart, which scales better and should be similarly instructive to non-experts.

2. Installation

DIMwid requires the packages Python, Qt, and PyQt. The minimal required (and tested) versions of these packages are listed in Table 1. On Linux-based systems (such as Ubuntu, Fedora, openSuse), these packages can normally be installed via the operating system's package manager. The installation under Windows requires the manual download and the execution of the package installers, but the installation is straightforward. We note that recent Windows packages for PyQt already contain the Qt framework, so Windows users only need to install Python and PyQt. The situation is similar for MacOS users. They only need to install a recent Python and a binary package for PyQt. We recommend PyQtX, whose complete installation includes a compatible version of Qt.

DIMwid itself is available on GitHub at

`https://github.com/RobinQrtz/DIMwid`

under the mit license, which allows *DIMwid* and its source code to be used freely for all purposes. *DIMwid* does not need to be installed or prepared since it only consists of three Python source files. However, to successfully use it we need the trace files of the decoders inside the Moses machine translation framework.

3. Usage

3.1. Obtaining the Input

DIMwid can process all major output formats that are produced by the decoders inside the Moses framework and 2 new custom formats:

- the full decoder trace of the chart-based decoder (-Tall option) and
- the full decoder trace of the chart-based decoder of the multi bottom-up tree transducer extension (Braune et al., 2013).

A recent version of the (master) Moses framework is required for the -Tall option and the mbotdecoder branch of Moses is needed for the second custom format. We refer the reader to the technical documentation for the use of *DIMwid* in conjunction with the mbotdecoder, but next we recall how to obtain the required trace files for the standard decoders.

3.1.1. Standard Moses trace

DIMwid supports the standard Moses trace outputs for both the phrase-based stack decoder (Koehn et al., 2003), via the "Phrase" format, and the syntax-based chart decoder (Chiang, 2007; Hoang et al., 2009) for hierarchical and tree-based models, via the "Syntax" format. By default, these traces are obtained by calling the decoders with the `-t` (phrase-based) and `-T` (syntax-based) flags as in:

```
cat input | moses -f moses.ini -t > out
cat input | moses_chart -f moses.ini -T trace.log > out
```

These traces only contain information about the best translation for each input sentence and are thus reasonably small. They allow us to reconstruct how the reported translations were obtained from the rules and the input sentences.

3.1.2. Full stack and chart trace

Sometimes the information about the reported best translations is not sufficient for a successful analysis. The full stack or chart trace records much more information about the decoding process. It contains all items that are present in the stacks (used in phrase-based decoding) or the chart cells (used in syntax-based decoding). Consequently, it allows us to effectively inspect which hypotheses were considered by the decoder and to investigate the competing hypotheses. Not surprisingly, those traces tend to be huge.

For phrase-based decoding, the stack trace is obtained by enabling level 3 verbosity and logging the error output. This is typically achieved by:

```
cat input | moses -f moses.ini -v 3 2> trace.log > out
```

Unfortunately, running Moses with multiple worker threads (option `-threads`) ruins the desired output, since the outputs are not synchronized. Consequently, the

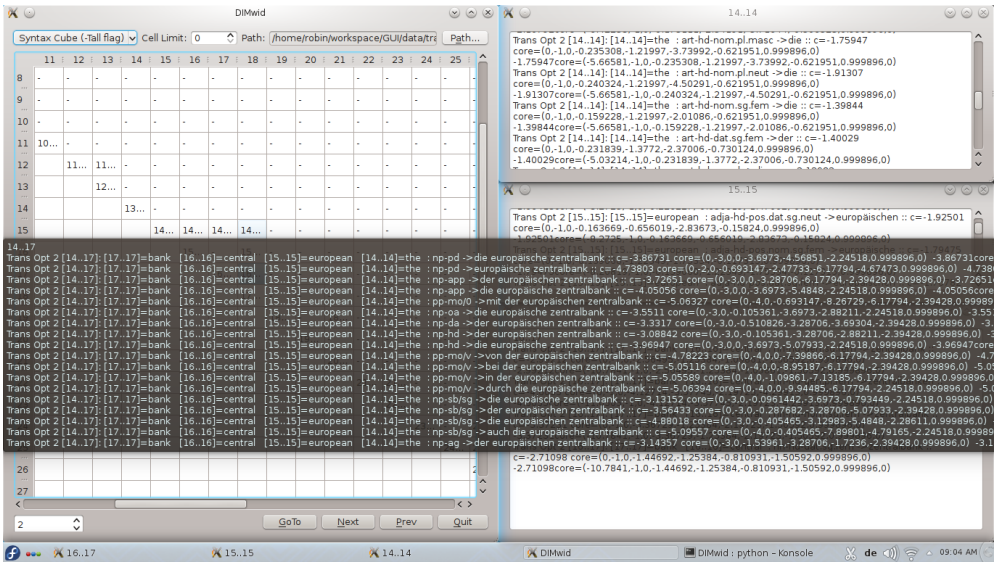

Figure 1. DIMwid presenting the chart and several details of the syntax-based decoder (display: KDE*).*

`threads`-option should not be present (neither in the call nor in the initialization file `moses.ini`) or explicitly set to "`-threads 1`". The obtained trace contains all translation options and recombination and stack information organized in a standard parse chart based on the covered spans. Since the trace is typically huge, it may take *DIMwid* a while to load it.

Alternatively, Moses also offers a flag named `-output-search-graph`, which outputs the entire search-space for the translation. This flag works for the phrase-based stack decoder and the syntax-based chart decoder. Since the output formats are different, the user needs to select the correct input format:

- "Phrase Stack (search-graph)" for the stack decoder or
- "Syntax Cube (search-graph)" for the chart decoder

in *DIMwid* when importing these traces.

For the syntax-based chart decoder we are also interested in the source-side of the used rules. This information is not provided in any of the existing formats, so we added a new output trace to Moses, which delivers also this information. The new flag is called `-Tall` and is used in the same way as the `-T` flag. A typical call (with some advanced options) might be:

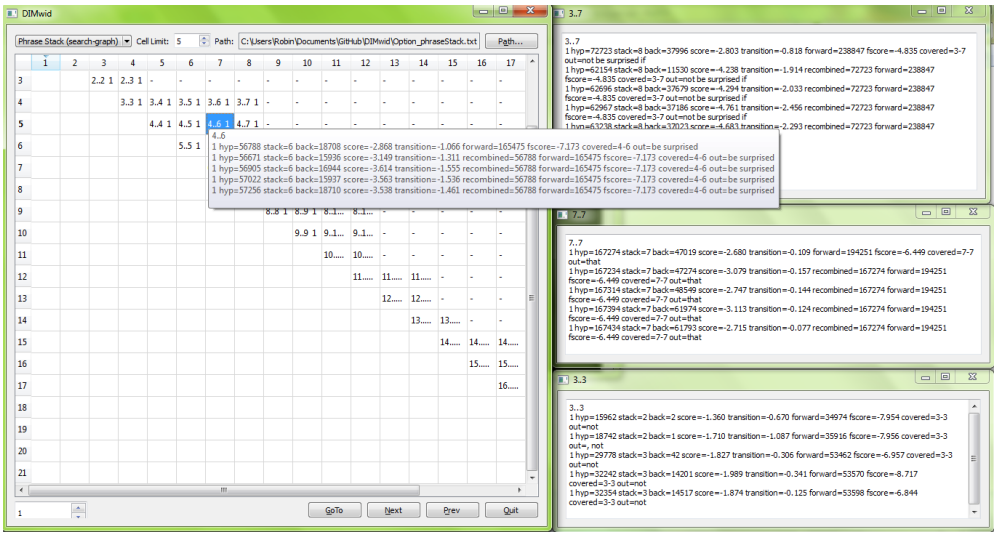

Figure 2. Display of the search-graph of a phrase-based decoder (display: Windows*).*

```
cat input | moses_chart -f moses.ini
   -include-lhs-in-search-graph -n-best-list listfile 100
   -T trace.log -Tall traceAll.log > out
```

The `-Tall` flag triggers the desired output, while the other flags `-n-best-list` and `-include-lhs-in-search-graph` produce more translation options and include the left-hand-sides of the rules. However, the output of the chart trace triggered by the `-T` and `-Tall` flags can be surprising. Due to Moses' internal decoding process the source-side nonterminal gets projected onto the corresponding nonterminal of the target-side. Thus, the reported source-side nonterminal might be different from the actual source-side nonterminal in the used rule, which should be considered when identifying the responsible rule. Nevertheless, the `-Tall` trace offers important information and prints — in contrast to the search graph — the correct terminals of the source-side.

3.2. Graphical User Interface

DIMwid is started by running `DIMwid.py`. The resulting main window behaves like any other window of the operating system and can therefore be maximized, minimized, moved around, etc. Keyboard commands are triggered by holding the keyboard's "Alt" key plus the underlined letter of the button. Next, we show the general steps needed to display a decoder trace:

1. First, we select the correct trace format by clicking on the "Format" button, which will open a drop-down menu containing buttons labelled with the supported formats. Once a format is selected, it will be shown on the button (instead of "Format").
2. Optionally, we may want to limit the number of items per chart cell using the text-field next to the "Cell Limit" label. Unless the complete stack or chart information is essential, a reasonable bound on the displayed items is generally recommended because *DIMwid* tends to run slow when huge numbers of items need to be displayed.
3. Next, we select the trace file by clicking on the "Path" button. The standard file selection dialog of the operating system will open and will allow the selection of the desired file. Currently, *DIMwid* does not auto-detect the trace format, so if the file format does not correspond to the selection in Step 1, then an error message is displayed.
4. Once *DIMwid* finishes loading the trace file, the chart display is available for each input sentence. Automatically, the chart for the first input sentence, numbered 0, is displayed. Other input sentences can be selected directly by entering their number in the input box next to the "GoTo" button and pressing that button. Alternatively, the "Prev" and "Next" buttons can be used to cycle through the input sentences.
5. The content of each chart cell is explained in Section 3.3. It can be shown in two ways: (a) A tool-tip window preview of the chart cell content is displayed when moving the mouse cursor over the chart cell. (b) A double-click on the chart cell opens a new window displaying the full cell content. Multiple windows allow the comfortable comparison of the contents of several cells.

3.3. Chart Display

Once the trace is successfully loaded, a quadratic table is displayed with a blank lower left triangle (see Figure 2). It has as many rows and columns as there are words in the selected source sentence. Each chart cell corresponds to a span in the input sentence. Its row marks the beginning of the span and its column marks the end of the span. Correspondingly, the entry (m, n) contains the translations of the span $[m+1, n+1]$ of the source sentence, which starts at the $(m+1)^{\text{th}}$ word and ends at the $(n+1)^{\text{th}}$ word. The diagonal (where $m = n$) contains the single word translations and the rightmost cell at the top (the cell $(0, 5)$ in Figure 2) contains the translations for the whole sentence. Spans for which no items are available are marked with a dash "-".

As a simple example, suppose that the decoder translated

```
Bob sends Alice a secret message
```

from English into the German sentence

```
Bob sendet Alice eine Geheimbotschaft
```

	0	1	2	3	4	5
0	Bob	-	-	-	-	-
1		sendet	-	-	-	-
2			Alice	-	-	-
3				eine	-	-
4					geheime	Geheimbotschaft
5						Nachricht

Table 2. A trivial example chart illustrating DIMwid*'s chart display.*

Table 2 shows a few (made-up) entries in the chart display for that translation. We omitted items for clarity. In the actual display the full translation should occur in cell $(0, 5)$. However, Table 2 shows that *"secret message"*, the span $[5, 6]$ of the source sentence, can be translated to *"Geheimbotschaft"*. Therefore the cell $(4, 5)$ of the chart contains this translation. In addition, the decoder could also use the translation into *"geheime Nachricht"* using the translations of cells $(4, 4)$ and $(5, 5)$.

The actual content of the items in a cell differs based on the trace format. For example, the items contain the source side in the level 3 verbosity output for phrase-based stack decoding. The Moses syntax-based chart decoder typically only outputs the terminals and the nonterminals of the target-side of the rule. As a minimum, the span of the source, which is translated, and its translation are always shown. The additional information depends on the trace format and is very specific and confusing to non-expert users of Moses. We decided to preserve this information for expert users, but most users can probably ignore the additional information. As illustrated in Section 4, it is very simple to adjust this behavior to accommodate any special needs.

4. Development

As mentioned earlier, we selected Python and Qt with straightforward adjustability in mind. Python code is usually easy to read, runs on all major operating systems, and is very common in the programming community. The graphical framework Qt is also freely available for all major operating systems, very common, and has Python bindings, which allows us to exclusively use Python for *DIMwid*.

4.1. Structure

DIMwid consists of three Python source files. `DIMwid.py` creates the application using the interface designed in `DIMterface.py`, which sets up the classes related to the graphical user interface. Finally, `DIMputs.py` contains all the classes that represent the different input formats and provides functions for reading those formats.

4.2. Hacking

The simplicity of *DIMwid* allows for quick changes in the code. It can easily be adjusted to display any kind of output sorted into spans. No inside knowledge about Python or the Qt framework is required. Let us illustrate this by showing the steps needed to support a new input format. First, we add a class to `DIMputs.py` for the text-format that we want to load. We can follow the example of the other classes in `DIMputs.py`. In order to use our new class, the `DataInput`-class needs a function which reads the new format and stores the contained information into an object of the newly created class. At this point the basic functionality is present, but we still need to enable the new format in the graphical user interface. This is achieved by the following steps:

1. add a new format button to the *Format-Buttons-Drop-Down-Menu*,
2. create a new *WidgetAction* corresponding to this button,
3. connect the *WidgetAction* with the drop-down menu,
4. create a function that sets the MainWindow's format to the new format,
5. connect the button to the format-setting function, and
6. add the new format to the `setPath`-function's *if-else*-block.

Since these code blocks exist for the natively supported formats, even non-experts should be able to perform these changes with the help of simple copy & paste actions.

5. Conclusion and Future Work

Our primary goal during the development of *DIMwid* was to make the analysis of the translation process easier. Such an analysis is beneficial for translation engineers that want to improve their system and to instructors that want to demonstrate the workings of the decoders in class. The graphical user interface of *DIMwid* displays the information of the typically huge and hardly readable trace files in an accessible manner. Currently, *DIMwid* supports (all) the trace outputs of both the phrase-based stack decoder and the syntax-based chart decoder of the Moses framework. The trace is uniformly presented in a chart, so all reported information is associated to a chart cell based on the covered span. Although *DIMwid* was developed for the traces of the Moses framework, it can easily be extended to read outputs of other frameworks (such as Joshua).

At present, *DIMwid* shows all items in the order that they occur in the traces. In future versions, we plan to combine the utility of the standard and full trace by highlighting the items that contribute to the best translation in the display of the full trace. In addition, we plan to add a format auto-detection that would remove the need to manually select a format. Ideally, we would also be able to graphically link items to their constituting items (i.e., the subspan items that were combined to form the current item). However, this feature has to be carefully implemented as it requires addi-

tional processing of the trace file and can thus potentially lead to major slow-down of *DIMwid*.

Acknowledgements

All authors were financially supported by the German Research Foundation (DFG) grant MA / 4959 / 1-1.

Bibliography

Braune, Fabienne, Andreas Maletti, Daniel Quernheim, and Nina Seemann. Shallow local multi bottom-up tree transducers in statistical machine translation. In *Proc. ACL*, pages 811–821. Association for Computational Linguistics, 2013.

Chiang, David. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.

Hoang, Hieu, Philipp Koehn, and Adam Lopez. A uniform framework for phrase-based, hierarchical and syntax-based machine translation. In *Proc. IWSLT*, pages 152–159, 2009.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proc. NAACL*, pages 48–54. Association for Computational Linguistics, 2003.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proc. ACL*, pages 177–180. Association for Computational Linguistics, 2007. Demonstration session.

Li, Zhifei, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren N. G. Thornton, Jonathan Weese, and Omar F. Zaidan. Joshua: an open source toolkit for parsing-based machine translation. In *Proc. WMT*, pages 135–139. Association for Computational Linguistics, 2009.

Weese, Jonathan and Chris Callison-Burch. Visualizing data structures in parsing-based machine translation. *The Prague Bulletin of Mathematical Linguistics*, 93:127–136, 2009.

Address for correspondence:
Robin Kurtz
kurtzrn@ims.uni-stuttgart.de
Universität Stuttgart
Institut für Maschinelle Sprachverarbeitung
Pfaffenwaldring 5b, D-70569 Stuttgart, Germany