Dialog State Tracking Challenge 2 & 3

http://camdial.org/~mh521/dstc/

Matthew Henderson, Blaise Thomson and Jason Williams

September 2013

Contents

1 Motivation

The term 'dialog state' loosely denotes a full representation of what the user wants at any point in a dialog. An effective dialog system must include a tracker which is able to accurately accumulate evidence over the sequence of a dialog, and adjust the dialog state according to the observations.

In the resarch community, the sharing of dialog corpora labelled with dialog state is a valuable resource to advance the state-of-the-art. Without such a corpora, different groups usually use data from disparate domains, precluding direct comparisons. The availability of a common dialog state corpora also opens the problem to new researchers and smaller research groups in the field; developing a whole dialog system then collecting the large number of dialogs required is expensive and time-consuming.

In 2013, the first dialog state tracking challenge (DSTC 1) released such corpora [3].[1] The domain concerned route information for buses in Pittsburgh, also called *Let's Go* [1]. Here the dialog state was the user's preferences for a variety of slots (route number, to and from locations, time of day etc.)

[1]The DSTC1 data remains available at http://research.microsoft.com/en-us/events/dstc/

Here we propose follow-up challenges, DSTC 2 and DSTC 3. The task differs from the Let's Go domain (the domain of the first DSTC) in several interesting ways. The most important is that it requires a more dynamic dialog state; while in DSTC the user goal was fixed and unchanging, users are allowed to change their mind in the restaurant domain.

The corpora for DSTC 2 and 3 were collected using Amazon Mechanical Turk, and consist of dialogs in two domains: restaurant information, and tourist information. Tourist information subsumes restaurant information, including bars, cafés etc. as well as multiple new slots. There will be two rounds of evaluation using this data: **DSTC 2** will release a large number of training dialogs related to restaurant search. Compared to DSTC (which was in the bus timetables domain), DSTC 2 will introduce changing user goals, tracking 'requested slots' as well as the new restaurants domain. Results from DSTC 2 will be presented at SIGDIAL 2014.

DSTC 3 will address the problem of adapation to a new domain, tourist information. DSTC 3 will release a small amount of labelled data in the tourist information domain; participants will use this data plus the restaurant data from DSTC 2 for training. Results from DSTC 3 will be presented at SLT 2014 (tentative).

The following section gives details of how to participate in the research challenge; section 4 details the evaluation procedure; section 3 then describes the corpora of dialogs being used in the challenge; lastly section 6 explains the labelling procedure.

2 Participation

In this challenge, participants will be provided with labelled human-computer dialogs to develop dialog state tracking algorithms. Algorithms will then be evaluated on a common set of held-out dialogs, offline, to enable comparisons. Participants will not need to implement or operate a speech recognizer, parser, real-time system, or text-to-speech engine.

As well as a corpus of labelled dialogs, participants will be given code that implements the evaluation measurements and a baseline tracker.

There will be two evaluation rounds that participants may enter, the first targeting SIGDIAL 2014 (DSTC 2) and the second SLT 2014 (DSTC 3). For evaluations, participants will be given an unlabelled test set, and a week in which to run their algorithm. Participants will submit their trackers' output to the organizers, who will then perform the evaluation. After evaluation, the test set transcriptions and labels will be made public. The scoring results and submitted output of all participants will also be made public. If making your tracker's output public would prevent your team from participating, please contact the organizers.

Participants are provided with:

1. Training data: Annotated system log files in a common JSON format
2. The scoring tool that will be used to evaluate the trackers
3. A baseline state tracker

2.1 Rules

Participation is welcomed from any research team (academic, corporate, non-profit, government). Members of the organizational committee and advisory committee are permitted to participate. In general, the identity of participants will not be published or made public. In written results, teams will be identified as team1, team2, etc. There are 2 exceptions to this: (1) the organizers will verbally indicate the identities of all teams at the conference/workshop chosen for communicating results; and (2) participants may identify their own team label (e.g. team5), in publications or presentations, if they desire, but may not identify the identities of other teams.

On submission of results, teams will be required to fill in a questionnaire which gives some broad details about the approach they adopted.

In particular, there is interest in writing trackers that would be feasible for use in live systems. We provide a few recommendations for participants to follow so trackers may be directly comparable to others:

- A tracker should be able to run fast enough to be used in a real-time dialog system
- A tracker should not make multiple runs over the test set before outputting its results
- Similarly, a tracker should not use information from the future of a dialog to inform its output at a given turn

These are not enforced, but will be addressed in the questionnaire at evaluation time.

2.2 Schedule

The schedule below is split into the two evaluation rounds, which will run one after the other. These are referred to as DSTC 2 and DSTC 3 respectively.

DSTC 2 uses only data from the restaurant information domain, while DSTC 3 introduces the tourist information domain with only a few unlabelled dialogs for the development stage. Thus DSTC 2 tests models ability to deal with changing goals, the richer dialog state as well as the new domain relative to DSTC 1. DSTC 3 will assess a tracker's ability to adapt quickly on-line to a larger domain. For more details on the datasets, see section 3.

Announcements and discussion about the challenge will be conducted on the group mailing list. Participants should be on the mailing list. Instructions for joining can be found on the DSTC homepage:
`http://camdial.org/~mh521/dstc`

3 Data

This section describes the data being released for DSTC 2 and 3. Datasets consist of dialogs, each one in its own directory. Every dialog has a '`log.json`' file, containing a Log Object. Labelled datasets also include a '`label.json`' file, a Label Object which annotates the dialog. The user's true utterance and action are annotated, as well as the dialog state. The domain of a dataset is described by an ontology object, also distributed in JSON. The formats of these objects are fully specified in Appendix A.

Date	**Description**
DSTC 2	
October 2013	Labelled restaurant information train and development sets released
20th January 2014	Unlabelled restaurant information test set released
27th January 2014	Tracker output on restaurant information test set due
3rd February 2014	Results on restaurant information test set given back to participants
5th March 2014	Approximate SIGDIAL deadline
DSTC 3	
4th April 2014	Labelled tourist information seed set released
9th June 2014	Unlabelled tourist information test set released
16th June 2014	Tracker output on tourist information test set due
23rd June 2014	Results on tourist information test set given back to participants
20th July 2014	Approximate SLT deadline

Table 1: Schedule

Slot	User may give as a constraint?
area	Yes, 5 possible values
food	Yes, 91 possible values
name	Yes, 113 possible values
pricerange	Yes, 3 possible values
addr	No
phone	No
postcode	No
signature	No

Table 2: Informable slots in DSTC2 (Restaurant Information Domain)

Slot	User may give as a constraint?
area	Yes, 15 possible values
children allowed	Yes, 2 possible values
food	Yes, 28 possible values
has internet	Yes, 2 possible values
has tv	Yes, 2 possible values
name	Yes, 163 possible values
near	Yes, 52 possible values
pricerange	Yes, 4 possible values
type	Yes, 3 possible values (restaurant, pub, coffeeshop)
addr	No
phone	No
postcode	No
price	No

Table 3: Informable slots in DSTC3 (Tourist Information Domain)

3.1 Informable and requestable slots

Each slot in DSTC2 and 3 is classified as "informable" or not. An "informable" slot is one which the user can provide a value for, to use as a constraint on their search. For example, the user can provide a value for the informable slot *pricerange*, but the system does not allow the user to specify a value for the non-informable slot *phone* (telephone number) – ie the system does not support looking up restaurants by phone number. Table 2 indicates which slots are informable in DSTC2, and table 3 indicates which slots are informable in DSTC 3.

All slots are "requestable" – i.e., the user can request the value of any slot. For example, for a given restaurant, the user can request the value of the *phone* or *pricerange* slot.

3.2 DSTC 2

Table 2 gives some information on the ontology, of the tourist information domain. This is the domain of all dialogs in DSTC 2.

The data used in DSTC 2 comes from 6 conditions; all combinations of 3 dialog managers and 2 speech recognisers. There are roughly 500 dialogs in each of the 6 conditions, and the condition is specified in the `system-specific` field of the Log object (see Appendix A) using the numberings below.

The 3 dialog managers are:

- **0**, an MDP for tracking the dialog state, and a hand-crafted policy
- **1**, a POMDP dialog state tracking, and a hand-crafted policy
- **2**, a POMDP a policy learnt using reinforcement learning

The 2 speech recognisers are:

- **0**, GMM-HMM model with artificially degraded acoustic models
- **1**, full GMM-HMM model optimised for the domain

These give two acoustic conditions, the degraded model producing dialogs at higher error rates.

When a tracker is deployed, it will inevitably alter the performance of the dialog system it is part of, relative to any previously collected dialogs. In order to simulate this, and to penalise overfitting to known conditions, testing dialogs in DSTC 2 are drawn from a dialog manager condition which is not found in the labelled training data. The set of all calls with dialog manager 2, with both speech recognition configurations, constitutes the test set. All calls with the other two dialog managers are used for the training and development set. Specifically, the datasets are arranged as so:

- **dstc2_train**. Labelled dataset released in October 2013, with 1612 calls from dialog managers 0 & 1 and speech recognisers 0 & 1.
- **dstc2_dev**. Labelled dataset released at the same time as dstc2_train, with 506 calls from dialog managers 0 & 1 and speech recognisers 0 & 1. No caller in this set appears in dstc2_train.
- **dstc2_test**. Set used for evaluation. Released unlabelled at the beginning of the evaluation week. After the challenge has completed, the labels will be released. This consists of all 1117 dialogs with dialog manager 2.

3.3 DSTC 3

The dialogs released for DSTC 3 are in a more complex domain than DSTC 2, referred to as the Tourist Information Domain. The ontology has more slots, and effectively extends that of DSTC 2. Details of the Tourist Information domain are given in Table 3.

DSTC 3 focuses on the challenge of extending the domain of a dialog system. The entire labelled datasets from DSTC 2 will be available, as well as a small set of unlabelled dialogs in the new domain. The two datasets used in DSTC 3 are:

- **dstc3_seed** 10 labelled dialogs released prior to the evaluation period
- **dstc3_test** 2265 dialogs used for evaluation.

The calls come from a mixture of 4 different dialog managers (all in the same acoustic condition). Again, the labels will be released after the evaluation has been run.

4 Evaluation

There is no single obvious metric to optimise in dialog state tracking, so the approach taken is to facilitate computation of multiple metrics using different configurations. In DSTC 2 and 3, we draw attention to a small selection of metrics of particular interest, referred to as 'featured metrics'. The scoring script used for evaluation, written in Python, is packaged with the data (see section 5), and takes as input a tracker output object in JSON, according to the specification given in A.3.

4.1 Tracker output

A tracker outputs for each turn a set of distributions for each of the three components of the dialog state:

- **Goals**:
  - For each informable slot in the ontology, a distribution over the values for that slot. Any remaining probability mass is given to the hypothesis that the user has not yet mentioned this slot
  - A distribution over joint goals. If this is omitted, then it is assumed that the goal should be composed of independent combinations of the slot distributions. The evaluation script scores the reported joint distributions as well as the distributions calculated making this assumption for reference.
- **Method** - a distribution over methods. The list of possible values is given in the ontology distributed with the data.
- **Requested slots**- for each requestable slot in the ontology, a binary distribution over whether this slot has been requested by the user and the system should inform it.

For an example of labelling each of these components of the dialog state, see section 6. Each is described in some detail below.

4.1.1 Goals

The goal at a given turn is the user's true required value for each slot in the ontology as has been revealed in the dialog up until that turn. If a slot has not yet been informed by the user, then the goal for that slot is 'None'.

4.1.2 Method

The 'method' of a user's turn describes the way the user is trying to interact with the system, and is one of the following values: by name, by constraints, by alternatives or finished. The method may be inferred if we know the true user's action using the following rules:

1. The method becomes 'by constraints' if the user gives a constraint by specifying a goal for a particular slot. E.g. `inform(food=chinese)`
2. The method becomes 'by alternatives' if the user issues a 'reqalts' act.
3. The method becomes 'by name' if the user either informs a value for the name slot, i.e. the user requests information about a specific named venue.

4. The method changes to 'finished' if the user gives a 'bye' act.

(See A.5 for a breakdown of all user actions). It is possible that the user makes no action which triggers one of the above rules at the start of the dialog, so the method is initially labelled as "none".

4.1.3 Requested slots

As mentioned above, for each turn a tracker must output a score for each requestable slot which specifies its confidence that the user has requested this slot and the system should inform it. The set of true requested slots is accumulated throughout the dialog as follows:

1. At the start of the dialog, the set is initialised as empty
2. If the system informs the value of s then s is removed from the set (if it is a member), as it is no longer required by the user
3. If the user requests a slot s (action contains `request(s)`) then s is added to the set

4.2 Metrics

There are a variety of tracker performance that are measured, using the following metrics:

- **Accuracy** - Fraction of turns in which the tracker's 1-best hypothesis is correct. This measures raw 1-best accuracy.
- **Mean reciprocal rank** - Average of $1/R$, where R is the rank of the correct hypothesis
- **ROC performance** - This includes several metrics that assess the discrimination of the top hypothesis's score.
  - **Equal error rate** - The sum of false accepts (FAs) and false rejects (FRs) where FA=FR
  - **Correct accept 5** - Fraction of correct accepts (CAs) when there are at most 5% False Accepts (FAs)
  - **Correct accept 10** -Fraction of CAs when there are at most 10% FAs
  - **Correct accept 20** - Fraction of CAs when there are at most 20% FAs
- **L2 norm** The L2 norm between the distribution of scores output by the tracker and the label. A second version (binary L2) is calculated as $(1-p)^2 + q^2$ where p is the score given by the tracker to the correct hypothesis and q is the remaining probability mass. This is included because this is what was called 'L2' in the first DSTC.
- **Average probabilty** - This measures the average probability assigned to the correct item.
- **Log probabilty** - The average log probability given to the correct hypothesis. The logarithm is clipped at 10^{-5}.
- **Update precision and accuracy** - Measures the accuracy and precision of updates to the 1-best hypothesis from one turn to the next. For more details, see [2]. This paper motivates these metrics as they are highly correlated with dialog success in their data.

4.3 Evaluation Schemes

Each of these metrics is evaluated for the different components of the dialog state:

- **Goals** - for each slot separately, the joint distribution, and all slots' statistics combined
- **Method** - one true label, so not split into subcomponents
- **Requested slots** - for each requestable slot separately, and all slots combined

There are two 'schedules' which dictate which turns of the dialog to include:[2]

- **Schedule 1** - all turns are included
- **Schedule 2** - only turns included where there is some information about this state component in the dialog so far:
  - For the goal for a slot, this is when a value for the slot has been mentioned in the dialog so far either by the system or in the SLU list
  - For the method, this is when there is an SLU hypothesis so far which informs the method of the user
  - For whether a slot s is requested, this is when the SLU hypothesis `request(`s`)` has appeared. If the system informs the value of s, then this schedule is reset.

Lastly there are two labelling schemes:

- **Scheme A** - The state is accumulated forwards through the dialog. For example the goal for slot s is None until it is informed as $s = v$ by the user, at which point it becomes v until it is informed otherwise. This is the scheme used in the first DSTC.
- **Scheme B** - The state is accumulated backwards through the dialog. The label state is the next value the user settles on, and is reset in the case of goals if the slot value pair is given in a canthelp act by the system.

Consider an example to clarify the difference between Scheme A and B. If the sequence of informing the food slot by the user is {0:/, 1:/, 2:chinese, 3:/, 4:indian, 5:/ }, then the Scheme A labelling is: {0:/, 1:/, 2:chinese, 3:chinese, 4:indian, 5:indian}. The Scheme B labelling would be: {0:chinese, 1:chinese, 2:chinese, 3:chinese, 4:indian, 5:indian}, unless at the turn at index 3 the system gave `canthelp(food=chinese, ...)` in which case the Scheme B labelling is: {0:chinese, 1:chinese, 2:chinese, 3: indian:, 4: indian, 5:indian}. Note `canthelp`s don't affect the goal labelling in scheme A.

For methods if the sequence of user acts gives the following updates: {0:/, 1:byconstraints, 2:/, 3:finished}, then the Scheme A labelling is: {0:/, 1:byconstraints, 2:byconstraints, 3:finished}, and Scheme B is: {0:byconstraints, 1:byconstraints, 2:finished, 3:finished}.

Scheme B is defined for goals and methods, but does not make sense for requested slots. It is motivated by the fact that under Scheme A, a tracker is penalised for correctly identifying the user's goal before it is mentioned. Scheme

[2]DSTC1 included a third schedule (schedule3), which included only the last turn of the dialog. This has been dropped since in DSTC2 and 3, the user's goal often changes.

B may be used for comparing approaches which predict what the user will say they want before they say it.

The labels included in the data are computed under Scheme A, and a function is included in the scripts (`misc.LabelsB`) which can calculate the Scheme B labels.

4.4 Featured Metrics

All combinations of metrics, state components, schedules and labelling schemes give rise to hundreds of numbers to analyse. Although each may have its particular motivation, many of the metrics will be highly correlated. The results of the first DSTC [3] showed that the metrics roughly split into 3 independent groups; one measuring 1-best quality (e.g. accuracy), another measuring probability calibration (e.g. L2), and the last measuring discrimination (e.g. ROC metrics).

In DSTC 2 and 3 we select the following as featured metrics:

- **Accuracy** under schedule 2 and label scheme A
- **L2 norm** under schedule 2 and label scheme A
- **ROC CA 5** under schedule 2 and label scheme A

Each is calculated for **joint goals, method and combined requested slots**. This gives 9 numbers altogether which participants should focus on optimizing. Note however there will be no explicit ranking of the submitted systems in published results.

The report tool distributed creates a table with these featured metrics (see section 5).

5 Included Scripts and Tools

DSTC 2 and 3 comes with a download which packages some useful scripts and tools for dealing with the data.

5.1 Baseline Tracker

A baseline tracker is included with the data, just as with the first DSTC. The baseline tracker works in a simple deterministic manner. For each dialog state component (goal, method, requested) we consider the output of the SLU over the turns of the dialog to be providing evidence of the form:

$$slu_{t,\,c} \;:\; V_c \rightarrow \mathbb{R}$$

where t indexes over turns, c is the state component, V_c is the set of values for component c.

The state components c are the goals for each slot, the method and requested for each slot. For example consider the SLU n-best list:

- **0.6** : `request(phone)`
- **0.3** : `inform(area=east)`
- **0.1** : `bye()`

then the components c for which $slu_{t,\,c}$ has non-zero values are: 'method', 'requested phone' and 'goal area'. For all other components $slu_{t,\,c} = 0$. Altogether, the non-zero values are:

$$\begin{array}{rcl}
slu_{t,\,\text{method}}(\text{by constraints}) & = & 0.3 \\
slu_{t,\,\text{method}}(\text{by name}) & = & 0.6 \\
slu_{t,\,\text{method}}(\text{finished}) & = & 0.1 \\
slu_{t,\,\text{requested phone}}(\text{True}) & = & 0.6 \\
slu_{t,\,\text{goal area}}(\text{east}) & = & 0.3
\end{array}$$

For the goal components, the baseline tracker will output one hypothesis for each component g at turn t:

$$\arg\max_{v} \left(\max_{t' \leq t} slu_{t',\,g}(v) \right)$$

with score:

$$\max_{v} \left(\max_{t' \leq t} slu_{t',\,g}(v) \right)$$

The method and requested slots components are reported as the distribution given by $slu_{t,\,c}$ for that turn t, without using information from the previous turns.

This is an equivalent to the baseline tracker in the first DSTC; essentially doing no incorporation of sequential evidence, rather picking the best state so far.

There is a variant of the baseline tracker called the 'focus' tracker, which accumulates evidence and has a simple model of the state changing throughout the dialog. Here we define how it tracks goals and the method. Define for method and goal components c, the quantity $q_{c,\,t}$ as:

$$q_{c,\,t} = 1 - \sum_{v \in V_c} slu_{t,\,c}(v)$$

Note that we should have $0 \leq q_{c,\,t} \leq 1$ as the SLU list is normalised to sum to 1. For our example turn:

$$\begin{array}{rcl}
q_{t,\,\text{method}} & = & 0.0 \\
q_{t,\,\text{goal area}} & = & 0.7
\end{array}$$

This is interpreted as the probability that the SLU did not inform the component c. Recursively define $p_{c,\,t}(v)$ for each v in V_c as:

$$\begin{array}{rcl}
p_{c,\,t}(v) & = & slu_{t,\,c}(v) + q_{c,\,t} p_{c,\,t-1}(v) \\
p_{c,\,1}(v) & = & slu_{1,\,c}(v)
\end{array}$$

Thus if there is no evidence that c should be reset in the SLU, then $q_{c,\,t} = 1$, $slu_{t,\,c}(v) = 0 \ \forall v$ and $p_{c,\,t} = p_{c,\,t-1}$. If the context is sure that c is to be reset, then $q_{c,\,t} = 0$ and $p_{c,\,t} = slu_{t,\,c}(v)$. $p_{c,\,t}(v)$ may be interpreted as the probability that v is the most recently mentioned value for c, when trusting the distributions from SLU directly. The focus baseline tracker uses the $p_{c,\,t}$ distributions for its output, and does something similar for requested slots- allowing for the system action to reset the scores when a slot is informed.

The source code for the baseline trackers is included in `baseline.py`, please look there for full details on the specifics of the implementations.

5.2 Running and Evaluating Baseline

This section serves as an introduction to using the baseline tracker and the evaluation scripts.

You should have a `scripts` directory with a `config` directory within it. The `config` directory contains the definitions of the datasets, e.g. `dstc2_dev.flist` which enumerates the calls in the development set of DSTC 2. It also contains the ontology objects. You can run the baseline tracker like so:

```
python scripts/baseline.py --dataset dstc2_dev
    --dataroot data --trackfile baseline.json
```

or for the focus tracker:

```
python scripts/baseline.py --dataset dstc2_dev
    --dataroot data --trackfile baseline.json --focus True
```

This will create a file `baseline.json` with a tracker output object. The structure and contents of the output can be checked using `check_track.py`:

```
python scripts/check_track.py --dataset dstc2_dev
    --dataroot data --ontology scripts/config/ontology_dstc2.json
    --trackfile baseline.json
```

This should output '`Found no errors, trackfile is valid`'. The checker is particularly useful for checking the tracker output on an unlabelled test set, before submitting it for evaluation in the challenge.

The evaluation script, `score.py` can be run on the tracker output like so:

```
python scripts/score.py --dataset dstc2_dev
    --dataroot data --trackfile baseline.json
    --ontology scripts/config/ontology_dstc2.json
    --scorefile baseline.score.csv
```

This creates a file `baseline.score.csv` which lists all the metrics described in 4:

```
$ head -4 baseline.score.csv
state_component, stat, schedule, label_scheme, N, result
goal.food, acc, 1, a, 3934, 0.7747839
goal.food, l2, 1, a, 3934, 0.4041261
goal.food, l2.binary, 1, a, 3934, 0.4239493
```

If this runs too slowly, then change the default number of bins used for the ROC calculations (10000) with the `--rocbins` flag.

Lastly we can use `report.py` to format the results:

```
python scripts/report.py --scorefile baseline.score.csv
```

This prints out some tables, including the featured metrics table:

```
                                        featured metrics
-----------------------------------------------------------------------------
             |  Joint Goals   |    Requested    |      Method    |
-----------------------------------------------------------------------------
Accuracy     |   0.6120959    |   0.8936170     |   0.8303156    |
l2           |   0.6318690    |   0.1743412     |   0.2657975    |
roc.v2_ca05  |   0.0000000    |   0.0000000     |   0.3367601    |
```

5.3 Other Tools

There are a few other scripts included which may be of use for participants:

- `dataset_walker.py` - a Python class which makes it easy to iterate through a dataset specified by file list (`.flist`) in `scripts/config`
- `score_slu.py` - a scoring script which implements a few metrics for evaluating the performance of an SLU component. This is not documented as part of the Dialog State Tracking challenge, but included as a bonus.
- `misc.py` - contains some useful functions, including one which outputs the labels for labelling scheme B.

6 Labelling

This section briefly explains the procedure used to generate the labels in the corpus. Figure 1 gives an actual example of how a simple dialog would be labelled under this procedure.

Firstly the user utterances are transcribed by employing crowd-sourced workers on the Amazon Mechanical Turk platform. Given a WAV file, they are employed to type what was said.

A simple Phoenix grammar is then run on the transcriptions, giving reasonably accurate annotations for the user dialog acts. These are then checked and corrected by hand. This constitutes the largest amount of work which cannot be done by crowd workers, as it requires expert knowledge.

A deterministic tracker is run on the dialog transcription. This tracker takes the true user's action and system action as fully observed context, and is able to produce the correct target labels for the dialog state.

Finally the corpus is checked using a script which ensures the data is in the correct format, nothing is missing, and runs some sanity checks.

1. A Mechanical Turker transcribes the user recordings in a dialog, giving the following dialog transcription:
   (a) (Hello welcome [...] how can I help?)
   "A cheap restaurant"
   (b) (Did you say expensive?)
   "Um yep okay"
   (c) (Okay, Oak Bistro is a nice restaurant in the expensive pricerange.)
   "What is the phone number?"
2. The Phoenix grammar gives the following results for the user dialog acts:
   (a) inform(pricerange=cheap)
   (b) null()
   (c) request(phone)
3. The manual correction of the dialog acts fixes the second turn from null() to affirm().
4. The deterministic tracker is run on the corrected semantics in conjunction with the system's actions to produce the following dialog state sequence:
   (a) goals: {pricerange=cheap}; requested-slots:{}, method: byconstraints
   (b) goals: {pricerange=expensive}; requested-slots:{}, method: byconstraints
   (c) goals: {pricerange=expensive, name=Oak Bistro}; requested-slots:{phone}, method: byname

Figure 1: Labelling a simple example dialog

A JSON Data Formats

The datasets are distributed as collections of calls, where each call has a `log.json` file containing a Log object in JSON, and possibly a `label.json` containing a Label object in JSON representing the annotations. Also distributed with the data is an Ontology JSON object, which describes the ontology/domain of the calls. The below sections describe the structure of the Log, Label and Ontology objects.

The JSON structures are specified by giving the field in a `bold type-face`. Fields which map to an object produce an indented list of the fields contained in that object. Fields mapping to lists of one type of object, will again have that object specified in an indented list surrounded by square brackets.

In order to make this clear, we present an example specification and object. The below is a specification to describe a book object:

- `title`: (string)
- `version`: (integer)
- `authors` : [
    - `name` (string)
    - `nationality` (string)

  ]
- `publisher` :
    - `name` (string)

An example JSON book object conforming to this specification would then be:

```
{
    "title": "Proofs from the Book",
    "version": 4,
    "authors":[
        {
            "name": "Martin Aigner",
            "nationality": "Austrian"
        },
        {
            "name":"Gunter Ziegler",
            "nationality":"German"
        }
    ],
    "publisher":{
        "name":"Springer"
    }
}
```

A.1 Log Objects

Every call has a Log object saved in a file called `log.json`, following the below specification:

- `session-id`: a unique ID for this call (string)
- `session-data`: the date of the call, in yyyy-mm-dd format (string)

- `session-time`: the time the call was started, in hh:mm:ss format (string)
- `caller-id`: a unique ID for the caller (string)
- `system-specific`: an object with some information concerning the system used in this call, e.g. acoustic condition and dialog manager (see section 3).
- `turns`: a list of log-turn objects, which give the output of the system and the following response captured from the user. [
  - `turn-index`: the index of this turn, starting at 0 (integer)
  - `output`:
    - `transcript`: the text of the system prompt (string).
    - `dialog-acts`: the dialog-act representation of the system prompt, see section A.5 (object).
    - `start-time`: the time into the dialog this prompt started synthesising in seconds (float)
    - `end-time`: the time into the dialog this prompt stopped synthesising in seconds (float)
    - `aborted`: whether this prompt was aborted, because the system determined there was a barge-in from the user (boolean)
  - `input`:
    - `start-time`: the time into the dialog this user turn was determined to start (float)
    - `end-time`: the time into the dialog this user turn was determined to end (float)
    - `audio-file`: the filename of the recording on disk
    - `live`: the actual input the dialog manager received at the time of the call
      - `asr-hyps`: [
        - `asr-hyp`: a transcription hypothesised by the speech recogniser (string)
        - `score`: a score for the hypothesis, given by the recogniser. This is in log space, and computed as described in the ATK manual[3]. Note these will not necessarily sum to 1 when exponentiated. (float)

        ]
      - `slu-hyps`: [
        - `slu-hyp`: a hypothesised dialog act decoding of what the user said, conforming to the specification in section A.5.
        - `score`: the probability assigned to this hypothesis. These should sum to 1. (float)

        ]
    - `batch`: ASR results for this user turn, computed offline. Note this is only available for train and development, and will be released at the same time as the test labels.
      - `asr-hyps` : As above
      - `cnet`: Confusion network given by ASR, which is a list of sausages: [

[3]`http://mi.eng.cam.ac.uk/research/dialogue/ATK_Manual.pdf`

- `start`: Seconds into the user turn that this sausage starts (float)
- `end`: Seconds into the user turn that this sausage ends (float)
- `arcs`: The list of possible words in this sausage: [
  - `word`: the word this arc corresponds to. Special value "!null" corresponds to a null arc. (string)
  - `score`: the log probability of this arc (float)

  ]

]

]

A.2 Label Objects

Annotated calls have a Label object saved in a file called `label.json`, following the below specification:

- `session-id`: the unique ID of this call (string)
- `caller-id`: a unique ID for the caller (string)
- `turns`: a list of label-turn objects, which run in parallel to the Log objects turns, providing annotations on the turn level [
  - `turn-index`: the index of this turn, starting at 0 (integer)
  - `audio-file`: the filename of the recording on disk
  - `transcription`: a transcription of what the user said (string)
  - `semantics`: the true dialog-act representation of the user's utterance
  - `goal-labels`: the true user goal as an object mapping from slots to values. A slot not present has not been specified yet by the user. (object)
  - `method-label`: the true method (string)
  - `requested-slots`: the true slots requested by the user, as a list (list).

  ]
- `task-information`: an object giving information on the task as it was presented to the Mechanical Turk worker, and their feedback on the call
  - `goal`: information on the goal/task given to the worker
    - `text`: the text string given to the worker to describe their task.
    - **constraints**: a list of slot, value pairs, for the restaurant they must settle on (list)
    - `request-slots`: a list of slots the user must request from the system (list)
  - `feedback`: subjective results from the user
    - `success`: the subjective success of the dialog (boolean)
    - `comments`: the user's general comments about the call. (string)
    - `questionnaire`: A list of question, answer pairs (list)

A.3 Tracker Output Objects

- `dataset`: the name of the dataset over which the tracker has been run (string)

- `wall-time`: the time in seconds it took to run the tracker (float)
- `sessions`: a list of results corresponding to each call in the dataset, in order: [
    - `session-id`: the unique ID of this call
    - `turns`: [
        - `goal-labels`: an object whose keys are slots mapping to distributions over values. The distributions are themselves objects whose keys are the values, mapping to probabilities (floats). If a slot is missing, then it is assumed the full probability mass is on the hypothesis that this slot has not yet been mentioned by the user. Probabilities within distributions must sum to no more than 1, with the remainder given to the not-mentioned hypothesis. (object)
        - `goal-labels-joint`: a list of joint hypotheses, whose `"score"` values must sum to no more than 1. The remaining probability mass is assumed to be on the hypothesis that the user hasn't mentioned any slot yet: [
            - `score`: the reported probability for this hypothesis. (float)
            - `slots`: a mapping from informable slots to values. Omitting a slot corresponds to the hypothesis that the goal for this slot has not yet been established. (object)

          ]
        - `method-label`: a distribution over methods, given as a objects whose keys are possible methods mapping to probabilities (floats). The probabilities should add to 1. (object)
        - `requested-slots`: an object which gives the probability that each requestable slot is requested. The keys are the slots, and they map to the probability (float). If a slot is omitted, it is assumed the probability is reported as 0. (object)

      ]

  ]

A.4 Ontology Objects

An Ontology object describes the ontology (or domain) of a set of calls, in the following format:

- `requestable`: a list of slots which may be requested by the user (list)
- `method`: the list of allowed possible values for the method (list)
- `informable`: a mapping of slots to lists of possible values for each of the slots the user may inform (i.e. give as a constraint) (object)

A.5 Dialog Act Objects

A dialog act is a shallow representation of the semantics of either a user's utterance or the system's prompt. In the case of the system prompt, the dialog act is input to a Natural Language Generation component which generates text to be synthesised.

The dialog-act notation used in DSTC 2 and 3 closely matches that used in the first DSTC.

Dialog acts are formed as lists of smaller JSON objects with two fields, `act` and `slots`. The `act` field maps to a string which denotes a general speech action or 'act type', and the `slots` field maps to a list of slot, value pairs giving some content for the act.

The below tables enumerate the possible act components for both user and system actions.

User Actions

`act` **field**	`slots` **field**	**Description**
`ack`	empty list	An acknowledgement e.g. "okay"
`affirm`	empty list	An affirmation e.g. "yes"
`bye`	empty list	Trying to end the dialog e.g. "good-bye"
`hello`	empty list	Greeting the system e.g. "hi"
`help`	empty list	Trying to solicit general help from the system e.g. "what can I say?"
`negate`	empty list	A negation e.g. "no"
`null`	empty list	Something not understandable to the system; outside its domain e.g. "pineapple"
`repeat`	empty list	A request for the system to repeat what it just said e.g. "please repeat that"
`reqalts`	empty list	Requesting for alternative suggestions e.g. "are there any others"
`reqmore`	empty list	Asking for more information in general e.g. "tell me more"
`restart`	empty list	Asking the system to start' from the beginning e.g. "let's start again"
`silence`	empty list	User actually said nothing
`thankyou`	empty list	User thanking the system e.g. "thanks"
`confirm`	one slot, value pair- (s, v)	s must be an informable slot and v a possible value for s as specified in the ontology. This corresponds to the user confirming that the constraint $s = v$ has been understood. E.g. "Is it in the west?" is a case with (s, v) = ("area", "west").

`deny`	one slot, value pair- (s, v)	s must be an informable slot and v a possible value for s as specified in the ontology. This is the user saying their goal for s is not v. E.g. “I don’t want something in the west”
`inform`	one slot, value pair- (s, v)	Again s must be an informable slot and v a possible value for s as specified in th ontology. This is the user specifying their goal for s as v. E.g. “It must be in the west”
`request`	one pair: (“slot”, s)	s must be requestable according to the ontology. This is the user asking for the value of s from the system E.g. “what part of town is it?”

Note that the set of possible slot, value pairs for `confirm`, `deny` and `inform` acts is determined by the ontology, which is shared with the data, and described in section 3. There is one special value for v in user acts, “dontcare”; (s, “dontcare”) is the case where no constraint should be given on the slot s. There is also a special act `{"act":"inform", "slots":[["this", "dontcare"]]}` which means the user has said ‘I dont care’ and the slot should be inferred from context.

Below are some examples of user dialog acts given in JSON, with corresponding transcriptions:

- `[{"act":"hello", "slots":[]},`
  `{"act":"inform", "slots":[["pricerange","cheap"]]}]`
  “Hello, I want something cheap.”
- `[{"act":"negate", "slots":[]},`
  `{"act":"deny", "slots":[["area","north"]]}`
  “No I don’t want it to be in the north.”
- `[{"act":"thankyou", "slots":[]},`
  `{"act":"bye", "slots":[]}`
  “Thank you good bye.”
- `[{"act":"ack", "slots":[]},`
  `{"act":"request", "slots":[["slot", "phone"]]}`
  “Okay, what’s the phone number of that place?”
- `[{"act":"inform", "slots":[["pricerange", "dontcare"]]}`
  “In any pricerange.”
- `[{"act":"inform", "slots":[["this", "dontcare"]]},`
  `{"act":"inform", "slots":[["food", "japanese"]]}`
  “I don’t mind, but it should serve japanese food.”
- `[{"act":"confirm", "slots":[["area", "centre"]]},`
  `{"act":"request", "slots":[["slot", "hastv"]]}`
  “Is it in the centre, and does it have television?”
- `[{"act":"reqalts", "slots":[]},`

```
{"act":"inform", "slots":[["type","pub"]]}
```
"Are there any other pubs?"

System Actions

`act` **field**	`slots` **field**	**Description**
`affirm`	empty list	An affirmation
`bye`	empty list	Good-bye message
`canthear`	empty list	Prompt the system uses if it detects a long period of silence
`confirm-domain`	empty list	In restaurant information domain, an action to confirm the user is indeed looking for a restaurant.
`negate`	empty list	A negation
`repeat`	empty list	Asking the user to repeat themselves
`reqmore`	empty list	Asking the user if they want any more information
`welcomemsg`	empty list	The message used to greet the user
`canthelp`	a non-empty list of slot, value pairs	Informing the user that the combination of constraints given in the list of slot, value pairs gives no matching entries in the database
`canthelp.missing_slot_value`	two pairs: ("slot", s) and ("name", v)	Informing the user that the value of slot s is not known for the venue named v
`expl-conf`	one pair (s, v)	An explicit confirmation from the system that the user's goal for s is v
`impl-conf`	one pair (s, v)	An implicit confirmation that the user's goal for s is v
`inform`	one pair (s, v)	Informing the value for slot s to be accepted as v
`offer`	one pair ("name", v)	Suggesting the venue named v to the user
`request`	one pair ("slot", s)	Asking the user what their goal for s is
`select`	one pair (s, v)	Always appears with at least one other select act with the same s but a different v. Asking the user to pick from the suggested values their goal for s
`welcomemsg`	empty list	The message used to greet the user

The slots and values for system actions are again drawn from the ontology.

There is one special slot, 'count' for inform actions (see below example).

Below are some example system actions, with corresponding prompts. Here to save space we compress the full JSON notation for acts:

- `request(food)`
  (JSON version: `[{"act":"request", "slots":[["slot","food"]]}]`)
  "What kind of food would you like"
- `inform(count=109), impl-conf(food=dontcare), request(pricerange)`
  "There are 109 restaurants if you don't care about the food. What pricerange would you like?"
- `canthelp(food=korean, pricerange=moderate)`
  "I'm sorry but there is no restaurant serving moderate korean food"
- `canthelp(food=spanish, pricerange=moderate), canthelp.exception(name="la tasca")`
  "I am sorry but la tasca is the only spanish restaurant in the moderate price range"
- `offer(name="yippee noodle bar"), inform(pricerange=moderate)`
  "Yippee noodle bar is in the moderate price range"
- `select(pricerange=cheap), select(pricerange=moderate)`
  "Sorry would you like something in the cheap price range or in the moderate price range"
- `offer(name="the lucky star"), inform(phone="01223 244277"), inform(addr= "cambridge leisure park clifton way cherry hinton")`
  "The phone number of the lucky star is 01223 244277 and it is on Cambridge Leisure Park Clifton Way Cherry Hinton"
- `repeat()`
  "Sorry I am a bit confused, please tell me again what you are looking for"
- `expl-confirm(area=centre)`
  "Let me confirm. You are looking for a venue in the central area"

References

[1] AW Black, S Burger, A Conkie, H Hastie, S Keizer, O Lemon, N Merigaud, G Parent, G Schubiner, B Thomson, JD Williams, K Yu, SJ Young, and M Eskenazi. Spoken dialog challenge 2010: Comparison of live and control test results. In *In Proceedings 12th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL), Portland, USA*, 2011.

[2] Ryuichiro Higashinaka, Noboru Miyazaki, Mikio Nakano, and Kiyoaki Aikawa. Evaluating discourse understanding in spoken dialogue systems. *ACM Transactions on Speech and Language Processing (TSLP)*, 1:1–20, 2004.

[3] Jason D Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. The dialog state tracking challenge. In *In Proceedings 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL), Metz, France*, 2013.