Inferring Ranked Dialog Flows from Human-to-Human Conversations

Javier Miguel Sastre Martínez, Aisling Nugent
Accenture The Dock, R&D Global Innovation Center,
7 Hanover Quay, Grand Canal Dock, Dublin, Ireland
{j.sastre.martinez, a.nugent}@accenture.com

Abstract

We present a novel technique to infer ranked dialog flows from human-to-human conversations that can be used as an initial conversation design or to analyze the complexities of the conversations in a call center. This technique aims to identify, for a given service, the most common sequences of questions and responses from the human agent. Multiple dialog flows for different ranges of top paths can be produced so they can be reviewed in rank order and be refined in successive iterations until additional flows have the desired level of detail. The system ingests historical conversations and efficiently condenses them into a weighted deterministic finite-state automaton, which is then used to export dialog flow designs that can be readily used by conversational agents. A proof-of-concept experiment was conducted with the MultiWoz data set, a sample output is presented and future directions are outlined.

1 Introduction

Virtual assistants are an attractive solution to customer service automation. While their language understanding capabilities and general knowledge of the world is limited in comparison with human agents, they can provide relatively simple services to an unlimited number of concurrent customers when coupled with cloud technologies. Additionally, they ensure an homogeneous experience, according to their programming. It is common practice to program a virtual assistant to fall back to a human agent whenever it detects it cannot provide a service, combining the strengths of both human and machine. Another usage of virtual assistants is to suggest to human agents a list of potential answers during a conversation with a customer, providing the agent potential useful information from previous interactions with the customer or from the customer profile, but letting the human decide the final answer.

Nowadays there exists a wide range of platforms for implementing virtual assistants, such as Google DialogFlow,[1] Amazon Lex,[2] Microsoft Bot Framework,[3] and RASA.[4] However, implementing a virtual assistant or extending it to support new services is not a trivial task. For the case of new services, one has to imagine how the conversations for that given service will be, or run a Wizard of Oz experiment with potential customers to gather examples of conversations. Once a conversational agent is deployed, it is often necessary to review its performance and adapt it to the actual conversations. For the case of services that are already being provided by human agents (e.g. in a call center), it is possible to review the conversation recordings in order to design a virtual assistant that will be better suited when first deployed. However, manually reviewing the call recordings can be time consuming.

In this paper we propose a technique to extract the most common workflows or dialog flows human agents follow when providing a specific service, once the calls are segregated by service.[5] The types of agent questions and responses are first identified and labeled (e.g.“Where are you going?” → “Destination request”), for which we use proprietary software. Once the dialog utterances are replaced by the labels, hundreds of conversation paths can be condensed and ranked in seconds as a weighted finite-state automaton. Different ranges of best paths in the automaton can then be exported as a succession of manageable-size dialog flows for their manual review (examples in the supplementary material). The conversational designer can then review them in rank order and decide when to stop, taking into account the added value of each successive dialog flow and the time available.

[1] https://cloud.google.com/dialogflow
[2] https://aws.amazon.com/lex/
[3] https://dev.botframework.com/
[4] https://rasa.com/
[5] A potential approach to segregation by service is discussed in Chatterjee and Sengupta (2020)

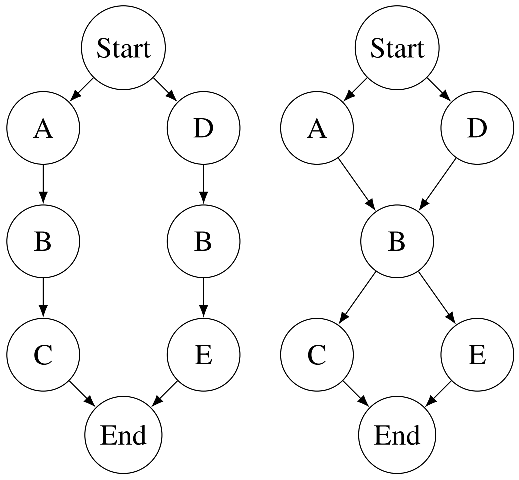


Figure 1: Example of 2 dialogs ABC and DBE (left) leading to overgeneration of sequences ABE and DBC (right) when only taking into account consecutive sequences of 2 dialog phases

2 Related Work

Bouraoui et al. (2019) present Graph2Bots, a tool that also aims to assist conversational agent designers. Similarly to us, they first identify types of utterances or dialog turns, which they call dialog phases. Then they build a graph with all possible dialog phases as nodes, and all possible transitions between consecutive dialog phases in the dialogs. Frequencies of dialog phases and transitions can then be used in order to filter out less frequent portions of the graph. We have also experimented with this kind of dialog phase graph and found several inconveniences we aim to overcome, namely

1. big convoluted graphs that, although they can be filtered, they are not partitioned so one can examine successive and manageable subsets of paths, one subset at a time,
2. the resulting graph represents concatenations of consecutive subsequences of 2 dialog phases from multiple dialogs, resulting in paths that do not actually exist in the dataset and produce confusion (Figure 1), and
3. the overgeneration of paths results in loops (Figure 2), which prevent the dialog flows from being loaded into conversational agent platforms as initial designs.

Qiu et al. (2020) propose an unsupervised approach to dialog structure inference based on a variational recurrent neural network with a structured attention layer that supports both 1 to 1 and

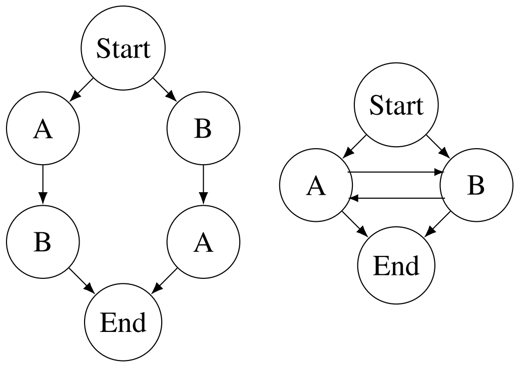


Figure 2: Example of 2 dialogs (left) leading to a loop (right) when only taking into account consecutive sequences of 2 dialog phases

multiparty conversations. However the reported times to train these models are in the order of hours, which in our use case would be impractical.

Zhai and Williams (2014) and Paul (2012) combine Hidden Markov Models and topic modeling to model the dialog structures as conversation states with probabilities to shift to other states, where each state models the potential language or topics in that state.

3 Rationale

This work builds on top of the output of a proprietary suite of tools for the analysis of call center conversations. This output comprises a set of dialog transcripts segregated by intent (e.g. booking a restaurant), where the utterances have been labeled by speaker role (agent/customer), classified into question/response/other, and then grouped into clusters of semantically equivalent questions/responses.[6] For each group of questions or responses, a canonical form of the question or response is provided to serve as the normalized version, analogous to the dialog phases in Bouraoui et al. (2019). Our goals are:

1. to find the most frequent sequences of questions and responses human agents follow, and
2. to compile them into a succession of dialog flows containing ranges of top-ranked sequences so that a conversational designer can visualize any number of them, starting from the highest ranked ones.

Also, as additional requirements,

[6]Other utterance types (e.g. greetings) are ignored

1. the paths in these dialog flows should come from actual dialogs and not be concatenations of subsequences from different dialogs (e.g. Figures 1 and 2), in order to avoid confusion and loops,

2. the size of individual dialog flows (number of paths) should be limited by means of a parameter, and

3. the dialog flows should also include some examples of potential customer utterances that may appear before and after each agent question or response so that one can determine the triggers of specific questions and responses as well as potential customer responses.

We do not intend to determine exact types of customer utterances but to provide a variety of examples since customer utterances tend to be more varied than agent utterances: whereas customers may request a service just a few times and may have no prior knowledge of the service protocols, agents deliver the same services multiple times to multiple customers and must adhere to established protocols and regulations.

4 Methodology

Overall, our proposed approach consists of 6 main steps:

1. building a non-deterministic finite-state automaton (NFA) representing all possible sequences of normalized agent questions and responses,

2. minimizing the NFA in order to obtain an equivalent but compact deterministic finite-state automaton (DFA),

3. annotating the DFA with question/response frequencies as well as with customer utterances

4. ranking the DFA paths and transitions and pruning it to a desired number of paths

5. selecting a maximum number of customer utterance examples before and after each agent utterance, discarding the rest, and

6. exporting consecutive ranges of ranked paths into separate dialog flows for their manual review.

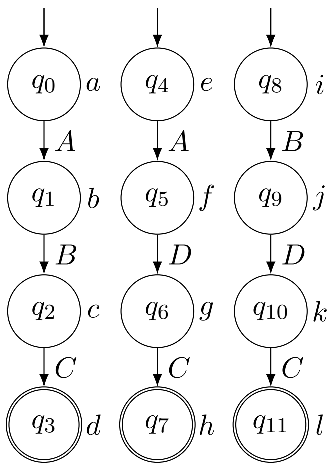


Figure 3: Example of NFA representing 3 dialogs $aAbBcCd, eAfDgCh$ and $iBjDkCl$

4.1 Building the NFA

For simplicity, let A, B, C and D be types of either agent questions or responses (their normalized versions or dialog phases). Let $a, b, \dots l$ be specific examples of customer utterances (non-normalized). We build an NFA as depicted in Figure 3, with a linear sequence of states (nodes) and transitions (edges) for each dialog, where transitions are annotated with the normalized agent utterances and states with the customer utterance examples. Note we only consider agent questions and responses, and other kinds of agent utterances are simply ignored (e.g. greetings). Consecutive sequences of customer utterances between 2 agent utterances are simply concatenated and treated as a single utterance. Consecutive sequences of agent utterances with no customer utterances in between result in a state that is annotated with no customer utterance (a state may have no customer utterance).

Formally, an NFA is defined as a 5-tuple $(Q, \Sigma, \delta, Q_I, F)$ with

- $Q = \{q_0, q_1, \dots, q_{|Q|-1}\}$, as a finite set of states,

- $\Sigma = \{\sigma_0, \sigma_1, \dots, \sigma_{|\Sigma|-1}\}$, as an either finite or potentially infinite input alphabet (normalized agent utterances in our case),

- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ as a finite and partial transition function where $\varepsilon \notin \Sigma$ is the empty symbol and $\mathcal{P}(\cdot)$ represents the set of all subsets of a given set,

- $Q_I \subseteq Q$ as the set of initial states (represented as nodes pointed by an arrow coming from nowhere), and

- $F \subseteq Q$ as the set of final states (represented as double-circled nodes).

A path in the automaton is an alternation of states and input symbols $q_i, \sigma_i, q_{i+1}, \sigma_{i+1}, \ldots$ starting and ending with a state, where for every subsequence $q_j, \sigma_j, q_j + 1$ there is a transition $\delta(q_j, \sigma) = q_{j+1}$. We say an automaton recognizes, represents or accepts an input sequence $\sigma_i \ldots \sigma_{i+n}$ iff there exists at least one path from an initial state to a final state with the same sequence of input symbols. We say an automaton is not deterministic iff it contains at least 2 paths starting from an initial state and labeled with the same sequence of input symbols (multiple states can be reached by consuming the same input sequence). Note having more than one initial state is sufficient for being non-deterministic.

We define the partial map $\zeta_c : Q \to \Gamma$ of states to customer utterances (Γ being the set of all customer utterances) to capture the customer utterance that may appear between 2 agent utterances, if any.

4.2 Minimizing the NFA

Minimizing an NFA results in an equivalent deterministic finite-state automaton (DFA) that represents the exact same set of input sequences but with a minimum set of states (see Figure 4). While this does not necessarily imply that the resulting automaton will have less transitions, this is usually the case for the NFAs that we build. Note for the sake of minimization, customer utterances are ignored (we only care about producing the same sequences of agent questions and responses). Formally, we define a DFA as a 5-tuple $(Q, \Sigma, \delta, q_I, F)$, where each element is defined in the same manner than for NFAs except for

- q_I, which is a unique initial state instead of a set of possible states, and
- $\delta : (Q \times \Sigma) \to Q$, which does not allow for empty symbols or more than one target state for the same source state and input symbol.

NFA minimization can be achieved by reversing the automaton, determinizing it, reversing it again and determinizing it a second time (van de Snepscheut, 1985). Reversing an automaton can be achieved by reversing the transitions, making initial states final, and final states initial. Since the NFAs we produce do not use empty input symbols, we can

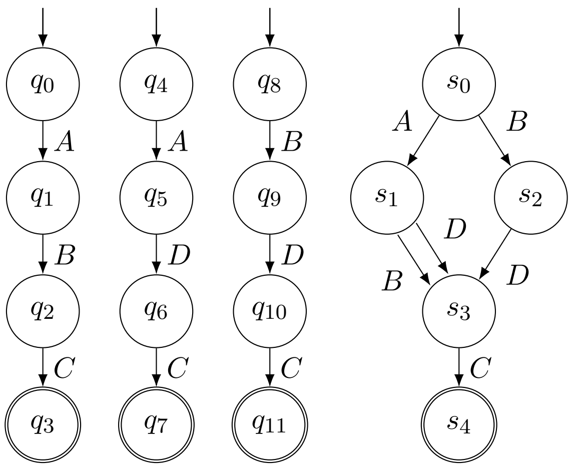


Figure 4: Example of NFA representing 3 dialogs (left) and DFA resulting from the NFA minimization (right)

use a simpler algorithm for determinizing them. Let A be one of these NFAs, Algorithm 1 (in the appendix) traverses all paths in A starting from its initial states, generating a DFA A' that contains a single state for each set of states that can be reached by consuming the same input sequence, and adding the corresponding transitions between the states in A'. It builds a map ζ_m of sets of states in A to states in A' to keep track of these correspondences and to avoid generating more than one state in A' for the same set of states in A. The algorithm starts by creating a single initial state q_I in A' corresponding to the set of initial states Q_I in A, and places the pair (Q_I, q_I) in a queue E of states to explore. As long as E is not empty, the next pair (Q_s, r_s) is dequeued and Algorithm 2 (in the appendix) is used to explore all the transitions coming from any state in Q_s, returning a map ζ_t of input symbols σ to sets of target states Q_t that can be reached from any state in Q_s by consuming σ. For each σ and Q_t, the corresponding state r_t in A' is either created or retrieved from ζ_m if already existed, and transition $\delta'(r_s, \sigma) = r_t$ is added to A'. Each time a state r_t is created for a given set of states Q_t, r_t is made final iff there is at least one final state in Q_t. Finally, whenever a new r_t is to be created due to the lack of a map $\zeta_m(Q_t)$, the map is added and (Q_t, r_t) is enqueued for further exploration of A.

4.3 Annotating the DFA

Let A be an NFA and A_{min} the resulting DFA upon minimization, since both machines are equivalent they recognize the exact same sequences. In the same way that during minimization we generate states of a DFA that correspond to sets of states in an NFA, there is a correspondence between states in A_{min} and states in A, as well as between transitions in A_{min} and transitions in A.

Given a map ζ_c of states in A to customer utterances (1 or none per state), Algorithm 3 (in the appendix) annotates the states in A_{min} with the sets of all customer utterances of the corresponding states in A (map ζ'_c), and annotates the transitions in A_{min} with the count of all equivalent transitions in A (map ζ'_f). An example is given in Figure 5. The algorithm also requires a topological sort of A_{min} as an input,[7] which can be computed with Kahn's (1962) algorithm. Algorithm 3 (in the appendix) explores both A and A_{min} synchronously, while computing the map ζ_m^{-1} of states in A_{min} to states in A. It starts by mapping the initial state of A_{min} to the set of initial states in A. Then explores the states of A_{min} by following the provided topological sort. For each state s_s in the sort, it retrieves the corresponding set of states $\zeta_m^{-1}(s_s) = Q_s$, and annotates s_s with the union of customer utterances in Q_s. Then for each transition $\delta'(\sigma, s_s) = s_t$ in A_{min} finds all the corresponding transitions $\delta(\sigma, q_s) = q_t$ in A, adding all the states q_t found to the mapping $\zeta_m^{-1}(s_t)$, and incrementing the count of transitions $\zeta'_f(s_s, \sigma, s_t)$ for each equivalent transition found in A. The topological sort is needed so that when exploring a next state s_s in the sort, we are sure the map $\zeta_m^{-1}(s_s)$ contains every possible corresponding state q_s in A, which will be the case since A and A_{min} are equivalent.

Apart from transition counts or frequencies, transitions of A_{min} can also be annotated with probabilities by normalizing the frequencies: for each set of transitions outgoing from the same source state, we compute the sum of frequencies of the transitions in the set, then divide the frequencies of these transitions by the sum. Log-probabilities can also be added in order to optimize the computation of top-scoring paths in the next section. A path score is the aggregation of the transition weights in the path, let it be the sum of frequencies, the product of probabilities, or the sum of log-probabilities.

4.4 Ranking and pruning

Given an annotated DFA A and a maximum desired number k of paths to keep (or carve), we use

[7] An ordering of all the states in A_{min} such that, for every transition in A_{min}, target states always come after source states in the ordering. This is the same problem as finding an ordering in a dependency graph.

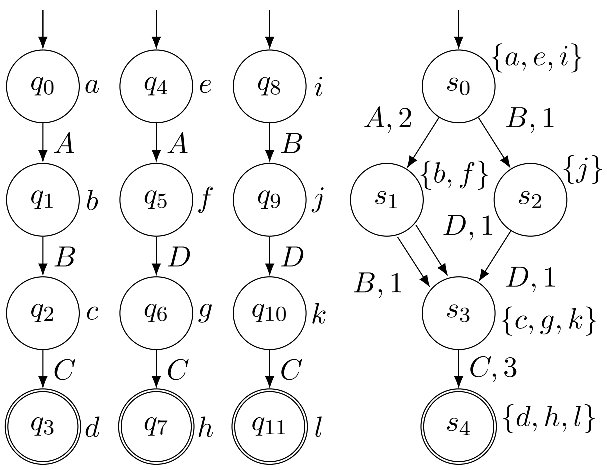


Figure 5: Example of NFA representing 3 dialogs (left) and equivalent DFA after state and transition annotation (right)

a Viterbi-like (1967) algorithm to efficiently compute the top-scoring paths, rank them (from 1^{st} to k^{th}) and annotate the transitions in A with the set of ranks of top paths they belong to. Transition rank annotations are used in the export step to generate the dialog flows for desired ranges of best paths. States and transitions that do not belong to any top-k path are removed in order to limit the execution time of the algorithm. Whereas this also limits the ranges of best paths that it will later be possible to export, in practice this limit can be much higher than the number of paths a conversational designer would deem necessary (e.g. 500), while keeping the execution time in the order of seconds. The algorithm is divided in 4 parts, which we detail in the following subsections: DFA preparation, forward propagation of weights, backward propagation of ranks, and DFA clean up. The first 3 parts also make use of a topological sort that is to be previously computed; the same topological sort used for the DFA annotation can be reused here. We use DFA in Figure 6 as an example. Note customer utterances are omitted since they are not relevant for the sake of ranking and pruning (the algorithms ignore map ζ'_c).

4.4.1 DFA preparation

The ranking and pruning algorithm computes the best paths between an initial and a final state of a DFA A. Whereas a DFA can only have a single initial state, it may have more than one final state. In order to take into account all possible paths in the DFA, we modify A as illustrated in Figure 7 so it contains a single and new final state s_f, with ε-

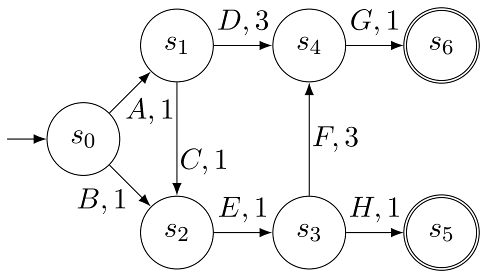


Figure 6: DFA with top-3 sequences ACEFG (1+1+1+3+1=7), BEFG (1+1+3+1=6) and ADG (1+3+1=5)

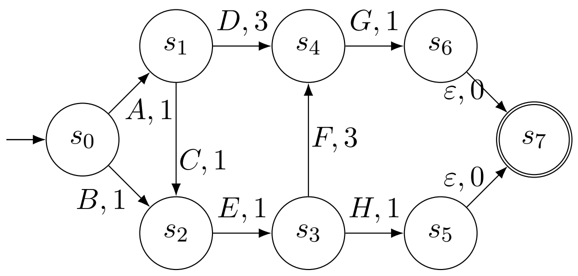


Figure 7: DFA after carving preparation

transitions arriving to it from each former final state and annotated with neutral weights (0 for frequencies and log-probabilities or 1 for probabilities). Strictly speaking, adding ε-transitions to A make it non-deterministic, however they will be removed during the clean up of A. Finally, the topological sort of A is to be updated by appending s_f at the end. In Figure 7, it ends up being $s_0, s_1, \ldots, s_7$.

4.4.2 Forward propagation of weights

For each state s_t in a carving-prepared DFA A, Algorithm 4 (in the appendix) computes the list L_t of k best possible aggregated weights that can be produced by reaching s_t from the initial state, and annotates s_t with this list (map $\zeta_L(s_t) = L_t$). An example of the computed lists is given in Figure 8 (lists above or below the states). Each element of L_t is a triplet (w, σ, s_s), with w being a top aggregated weight, and (σ, s_s) the symbol and source state of the previous transition that allowed for that best weight (transition $\delta(\sigma, s_s) = s_t$). The algorithm starts by initializing the lists L_t of all the states as empty lists. Then initializes $\zeta_L(q_I)$ with triplet $(w_{init}, \varepsilon, \perp)$, a initial aggregated weight (0 for frequencies and log probabilities, 1 for probabilities), and a non-transition (there is no transition before q_I). For each state s_s in the provided topological sort, except the last state s_f added during carving preparation, the algorithm propagates the corresponding top weights in $\zeta_L(s_s)$ towards the

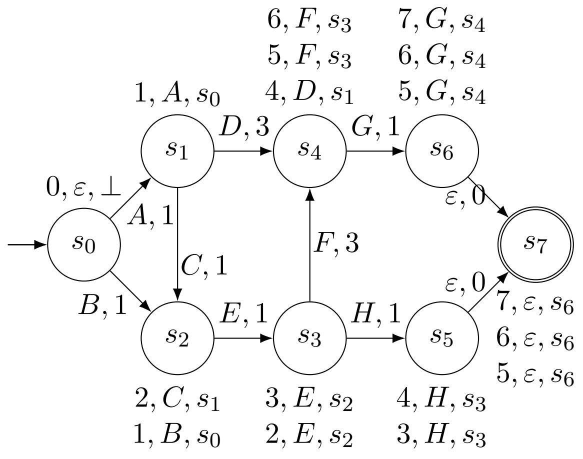
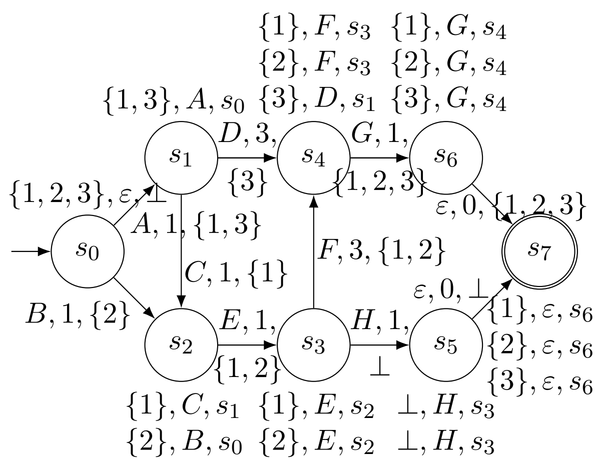


Figure 8: DFA after forward propagation

lists L_t of the corresponding target states s_t. Given a list L_s of $n \leq k$ elements, the n weights are combined with the weight of each transition from s_s, and the resulting aggregated weight is added to the list L_t of the corresponding target state s_t along with the corresponding transition symbol and source state s_s. Lists of top weights are sorted lists of at most k triplets, so when a list overflows the excess can be easily removed from its end. Thanks to the topological sort, whenever propagating the top weights of a state s_s we make sure all possible paths that reach s_s from q_I have been explored, and the list contains the top weights only (excess of weights will have been removed).

4.4.3 Backward propagation of ranks

Once the lists of top weights and last transitions have been computed, we can proceed to rank the final top weights in $\zeta_L(s_f)$ and propagate these ranks backwards, following the last transitions in the corresponding triplets of best weights. Algorithm 5 (in the appendix) starts by creating a list of sets of ranks for state s_f (map ζ_{SR} standing for *state ranks*), one set of ranks per triplet in $\zeta_L(s_f)$. The first set of ranks is $\{1\}$ (first rank), the second is $\{2\}$ (second rank), and so forth (see list below s_7 in Figure 9; we replaced top weights with rank sets to save space). Then these ranks are propagated backwards by following a reverse of the topological sort, excepting the initial state. Given a state s_t in the topological sort, the algorithm first computes a map ζ_{BR} (backwards ranks) of backwards transitions to the list of all possible sets of ranks in $\zeta_L(s_t)$. For instance, in Figure 9 the 3 top backwards transitions of s_f are the same, so ζ_{BR} contains in this case a single map $\zeta_{BR}(\varepsilon, s_6) = [\{1\}, \{2\}, \{3\}]$. For each



$\{1\}, F, s_3$ $\{1\}, G, s_4$
$\{2\}, F, s_3$ $\{2\}, G, s_4$
$\{1,3\}, A, s_0$ $\{3\}, D, s_1$ $\{3\}, G, s_4$
$D, 3,$
$G, 1,$
s_1
s_4
s_6
$\{3\}$
$\{1,2,3\}$
$\{1,2,3\}, \varepsilon, \perp$
$\varepsilon, 0, \{1,2,3\}$
$A, 1, \{1,3\}$
s_0
s_7
$C, 1, \{1\}$
$F, 3, \{1,2\}$
$\varepsilon, 0, \perp$
$B, 1, \{2\}$
$E, 1,$
$H, 1,$
s_2
s_3
s_5
$\{1\}, \varepsilon, s_6$
$\{2\}, \varepsilon, s_6$
$\{3\}, \varepsilon, s_6$
$\{1,2\}$
$\perp$
$\{1\}, C, s_1$ $\{1\}, E, s_2$ $\perp, H, s_3$
$\{2\}, B, s_0$ $\{2\}, E, s_2$ $\perp, H, s_3$

Figure 9: DFA after backward propagation

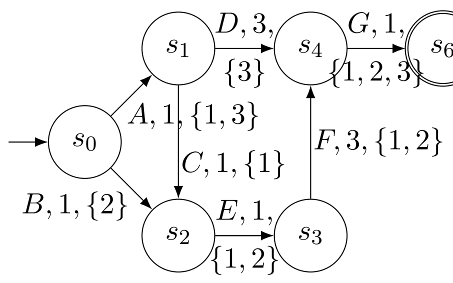


Figure 10: DFA after clean up

map $\zeta_{BR}(\sigma, s_s) = BR$, transition $\delta(s_s, \sigma) = s_t$ is annotated with the union of all the sets of ranks in BR (e.g. in Figure 9, transition $\delta(s_6, \varepsilon) = s_7$ gets ranks $\{1, 2, 3\}$. Map ζ_{TR} is used to annotate the transition ranks. Furthermore, for each list of sets of ranks $\zeta_{BR}(\sigma, s_s) = [R_1, R_2, \ldots]$, the list is propagated backwards towards $\zeta_{SR}(s_s)$ by computing the pairwise union of sets of ranks of $\zeta_{SR}(s_s)$ with $\zeta_{BR}(\sigma, s_s)$. For instance, in Figure 9 list $[1, 2, 3]$ below s_7 gets propagated as is to the list above s_6, since s_7 is the only contributor of ranks for s_6. States that get ranks are part of the top k paths and are marked as useful (states to be keep during clean up). For instance, no ranks get propagated to s_5 (symbol $\perp$ represents null), hence it will not be marked and will be removed during clean up. Ranks $[\{1\}, \{2\}]$ above state s_4 correspond to transition $\delta(s_3, F) = s_4$, hence get propagated to state s_3 (ranks below s_3). However rank $[\{3\}]$ of s_4 for transition $\delta(s_1, D) = s_4$ gets propagated to ranks of state s_1. Rank $[\{1\}]$ of s_2 for transition $\delta(s_1, C)$ also gets propagated to s_1, resulting in ranks $[\{1, 3\}]$ (pairwise union of sets of ranks). Once the algorithm ends, the ranks $[\{1\}, \{2\}, \{3\}]$ of s_7 have travelled back through the top paths, annotating the corresponding transitions and states, and arriving to state s_0 as $[\{1, 2, 3\}]$.

4.4.4 DFA clean up

Algorithm 6 (in the appendix) undoes the changes done to the DFA during carving preparation, and deletes every unmarked state (e.g. s_5) and unranked transition (e.g. $\delta(s3, H) = s_5$). The lists of state top weights and ranks are no longer needed and can be discarded; we just need to keep map ζ_{TR} of transition ranks. Figure 10 illustrates the resulting automaton for our example. In order to avoid potential data corruption, the algorithm deletes the states and transitions in a proper order, starting with transitions in $\zeta_L(s_f)$; these are ranked transitions but are added during carving preparation. Then states s_s are scanned in topological sort except for s_f. For each s_s, transitions from s_s with no ranks are removed. Then s_s is removed if it's not marked. Note that by following a topological sort, all transitions incoming to and outgoing from an unmarked state are removed before removing the state. Finally, s_f is removed unconditionally without scanning it, since it has no outgoing transitions and it was added during carving preparation. We no longer need the topological sort, so it can be discarded.

4.5 Selecting customer utterances

Due to the potential big number of customer utterances that might be annotated on the remaining DFA states, we want to select a limited number n of different examples per state and delete the rest so that the exported dialog flows are not overcrowded. For each set of customer utterances, we first compute the corresponding sentence embeddings (Cer et al., 2018; Reimers and Gurevych, 2019; Yang et al., 2020). Then we clusterize the sentences into groups of semantically similar ones using DBSCAN (Ester et al., 1996). We select the n biggest clusters and, for each one, we find the vector closest to the cluster centroids. Finally, we retrieve the sentences that correspond to those vectors, and delete all the rest.

4.6 Exporting dialog flows

Once the DFA is pruned, the transitions ranked, and the customer utterances filtered, generating a dialog flow for an arbitrary range of best paths is straightforward: we simply traverse the automaton starting from the initial state and following every transition that has at least one rank within the range, until no more states are found. Transition rank sets ζ_{TR} are sorted data structures (e.g. sorted lists or binary trees) so one can efficiently evaluate whether the intersection of the set with the range of ranks is empty or not. As states and transitions are traversed, the corresponding nodes and edges of the dialog flow can be exported to the desired format, e.g. DOT (Gansner and North, 2000) in order to create dialog flow visualizations, or some format of a conversational agent platform.

5 Methodology extension

An inconvenience of the method described above is that all the customer utterances that may start a conversation get grouped together in the DFA initial state (e.g. utterances a, e, and i of state s_0 in Figure 5). We would like to split this group into potential utterances that may precede each first agent utterance, so that we can also determine what triggered each first agent utterance. This can be achieved by modifying the way in which the NFA is built, as illustrated in Figure 11: we simply duplicate the first transition of each individual dialog, leaving the new initial states with no customer utterances. Upon minimization, the new first agent utterances will only allow for grouping the first customer utterances that are followed by the same agent utterance. Upon exporting the dialog flows, these first agent utterances are simply to be ignored.

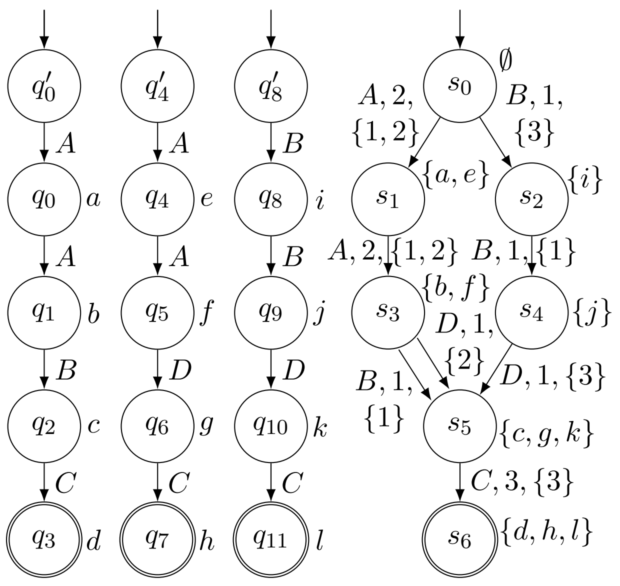


Figure 11: NFA with duplicated first transitions (left) and resulting NFA after minimization, annotation and ranking (right)

6 Results

We have tested this methodology with a sample of 492 restaurant booking dialogs from MultiWOZ (Han et al., 2020). On a MacBook Pro (2018), it took 4.3 seconds to run the extended method from NFA building (4083 states and 3591 transitions) to DFA ranking and pruning (1704 states, 2176 transitions) for a big enough k so all paths (488) were ranked and kept. Filtering the customer utterances took 36.8 additional seconds, though taking into account that this process included computing multilingual sentence embeddings (Yang et al., 2020) for all the customer utterances, it could be considerably reduced by using a GPU. Exporting a dialog flow of 50 paths into SVG with GraphViz (Gansner and North, 2000) took 2.5 seconds. Two flows are shown as supplementary material, and a wide range of flows has been provided as accompanying materials. Ranking criterion is frequency aggregation so longer paths are produced. For simplicity, only the top rank of each transition is shown. The process factors out prefixes and suffixes of agent utterance sequences, which to some extent allows for identifying the most common full sequences. The flows exactly reflect what is found in the data, which is what we initially intended.

7 Conclusion and future work

This paper presented a novel and efficient method for inferring ranked dialog flows from human-to-human conversations in seconds. This method converts the dialogs into summarised and digestible artefacts, in the form of weighted finite-state automata with ranked transitions. The method is intended to be used together with a semi-supervised iterative process of identification of types of agent utterances, hence the quick generation of the ranked dialog flows is a must.

Future work includes 1) splitting the customer bubbles across the entire dialog flows to have separate groups of examples of customer utterances before each agent bubble, 2) to identify dialog substructures such as subsequences of agent questions and responses that may appear in any order, so they can be replaced by a subautomaton call and allow for further path collapsing, and 3) to allow for a controlled amount of overgeneration/noise in the automaton that maximizes the number of collapsed paths (adding missing subsequences that allow for further minimization).

8 Acknowledgements

We thank Paul A. Walsh and the SIGDIAL 2022 reviewers for their feedback.

References

Jean-Leon Bouraoui, Sonia Le Meitour, Romain Carbou, Lina M. Rojas Barahona, and Vincent Lemaire. 2019. Graph2Bots, unsupervised assistance for designing chatbots. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 114–117, Stockholm, Sweden. Association for Computational Linguistics.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174, Brussels, Belgium. Association for Computational Linguistics.

Ajay Chatterjee and Shubhashis Sengupta. 2020. Intent mining from past conversations for conversational agent. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4140–4152, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press.

Emden R. Gansner and Stephen C. North. 2000. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30(11):1203—1233.

Ting Han, Ximing Liu, Ryuichi Takanobu, Yixin Lian, Chongxuan Huang, Dazhen Wan, Wei Peng, and Minlie Huang. 2020. Multiwoz 2.3: A multi-domain task-oriented dialogue dataset enhanced with annotation corrections and co-reference annotation. *arXiv preprint arXiv:2010.05594*.

Arthur B. Kahn. 1962. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562.

Michael J. Paul. 2012. Mixed membership Markov models for unsupervised conversation modeling. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 94–104, Jeju Island, Korea. Association for Computational Linguistics.

Liang Qiu, Yizhou Zhao, Weiyan Shi, Yuan Liang, Feng Shi, Tao Yuan, Zhou Yu, and Song-Chun Zhu. 2020. Structured attention for unsupervised dialogue structure induction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1889–1899, Online. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Jan L. A. van de Snepscheut. 1985. *Trace theory and VLSI design*, volume 200 of *Lecture Notes in Computer Science*. Springer-Verlag. PhD thesis, Eindhoven University of Technology.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-hsuan Sung, Brian Strope, and Ray Kurzweil. 2020. Multilingual universal sentence encoder for semantic retrieval. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 87–94, Online. Association for Computational Linguistics.

Ke Zhai and Jason D. Williams. 2014. Discovering latent structure in task-oriented dialogues. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 36–46, Baltimore, Maryland. Association for Computational Linguistics.

A Appendix

A.1 Algorithms

Algorithm 1 nfa_determinize(A)

```
Input: $A = (Q, \Sigma, \delta, Q_I, F)$, a NFA
Output: $A' = (Q', \Sigma, \delta', q_I, F')$, a DFA equivalent to $A$
1:  initialize $A'$ as a DFA with a single and initial state $q_I$ and no final states or transitions
2:  if $Q_I \cap F \neq \emptyset$ then
3:      $F' \leftarrow F' \cup \{q_I\}$
4:  end if
5:  $\zeta_m(Q_I) \leftarrow q_I$                ▷ state equivalence map
6:  $E \leftarrow \{(Q_I, q_I)\}$             ▷ equivalent-pairs queue
7:  while $E \neq \emptyset$ do
8:      $(Q_s, r_s) \leftarrow \mathrm{dequeue}(E)$
9:      $\zeta_t \leftarrow \mathrm{nfa\_recognize\_every\_symbol}(Q_s)$
10:     for each $(\sigma, Q_t) : \zeta_t(\sigma) = Q_t$ do
11:         $r_t \leftarrow \zeta_m(Q_t)$
12:         if $r_t = \perp$ then
13:             make new state $r_t \in Q'$
14:             if $Q_t \cap F \neq \emptyset$ then
15:                 $F' = F' \cup \{r_t\}$
16:             end if
17:             $\zeta_m(Q_t) \leftarrow r_t$
18:             $E \leftarrow E \cup (Q_t, r_t)$
19:         end if
20:         $\delta'(r_s, \sigma) \leftarrow r_t$
21:     end for
22: end while
```

Algorithm 2 nfa_recognize_every_symbol(Q_s)

```
Input: $Q_s$, a source set of states
Output: $\zeta_t : \Sigma \rightarrow \mathcal{P}(Q)$, a map of input symbols to target sets of states such that $\zeta_t(\sigma) = \bigcup_{q_s \in Q_s} \delta(q_s, \sigma)$
1:  initialize $\zeta_t$ as an empty map of $\Sigma \rightarrow \mathcal{P}(Q)$
2:  for each $q_s \in Q_s$ do
3:      for each $(\sigma, q_t) : \delta(q_s, \sigma) = q_t$ do
4:          if $\zeta_t(\sigma) = \perp$ then
5:              $\zeta_t(\sigma) \leftarrow \emptyset$
6:          end if
7:          $\zeta_t(\sigma) \leftarrow \zeta_t(\sigma) \cup \{q_t\}$
8:      end for
9:  end for
```

Algorithm 3 dfa_annotate(A, A_{min}, ζ_c)

```
Input: $A = (Q, \Sigma, \delta, Q_I, F)$, a NFA
       $\zeta_c : Q \rightarrow C$, map of states in $A$ to customer utterances
       $A_{min} = (Q', \Sigma, \delta', q'_I, F')$, DFA result of minimizing $A$
       $A_{min\_sort} : (Q \times Q \times \ldots)$, a topological sort of $A_{min}$
Output: $\zeta'_c : Q' \rightarrow \mathcal{P}(C)$, map of $A_{min}$ states to sets of customer utterances
       $\zeta'_f : (Q' \times \Sigma \times Q') \rightarrow \mathbb{N}_0$, map of $A_{min}$ transitions to frequencies
1:  $\zeta_m^{-1}(q_I) \leftarrow Q_I$ ▷ Inverse equivalent state map
2:  for each $s_s \in A_{min\_sort}$ do
3:      $Q_s \leftarrow \zeta_m^{-1}(s_s)$
4:      $\zeta'_c(s_s) \leftarrow \bigcup_{q_s \in Q_s} \{\zeta_c(q_s)\}$
5:      for each $(\sigma, s_t) : \delta'(s_s, \sigma) = s_t)$ do
6:          if $\zeta_m^{-1}(s_t) = \perp$ then
7:              $\zeta_m^{-1}(s_t) \leftarrow \emptyset$
8:          end if
9:          $Q_t \leftarrow \bigcup_{q_s \in Q_s} \delta(q_s, \sigma)$
10:         $\zeta_m^{-1}(s_t) \leftarrow \zeta_m^{-1}(s_t) \cup Q_t$
11:         $\zeta'_f(s_s, \sigma, s_t) \leftarrow |Q_t|$
12:     end for
13: end for
```

Algorithm 4 dfa_carving_forward_prop(A, $A_{sort}, \varepsilon, \zeta_w, w_{init}, \bullet, \prec, k$)

```
Input: $A = (Q, \Sigma, \delta, q_I, F)$, carving-prep. DFA,
       $A_{sort} : Q^{|Q|-1}$, a topological sort of $A$
       $\varepsilon$, a special symbol not in $\Sigma$ to denote the empty input
       $\zeta_w : (Q \times \Sigma \cup \{\varepsilon\}) \times Q) \rightarrow W$, map of $A$ transitions to weights
       $w_{init}$, the initial weight
       $\bullet$, the weight aggregation operator
       $\prec$, the weight comparison operator
       $k$, the number of paths to carve
Output: $\zeta_L$, a map of states $s_t \in Q$ to sorted lists of triplets $(w, \sigma, s_s) \in (W \times (\Sigma \cup \{\varepsilon\}) \times Q)$, each representing a top-k best weight $w$ produced by reaching $s_t$ through a last transition $\delta(s_s, \sigma) = s_t$
1:  for each $s \in Q$ do
2:      $\zeta_L(s) \leftarrow$ empty list
3:  end for
4:  append $(w_{init}, \varepsilon, \perp)$ to $\zeta_L(q_I)$
```

```
 5: for each $s_s \in A_{sort}$ except last do
 6:     $L_s \leftarrow \zeta_L(s_s)$
 7:     for each $(\sigma, s_t) : \delta(s_s, \sigma) = s_t$ do
 8:         $w \leftarrow \zeta_w(s_s, \sigma, s_t)$
 9:         $L_t \leftarrow \zeta_L(s_t)$
10:         for each $(w_s, \sigma', s_b) \in L_s$ do
11:             $w_t \leftarrow w_s \bullet w$
12:             insert $(w_t, \sigma, s_s)$ in $L_t$ maintaining $\prec$ weight order
13:             if size of $L_t$ > k then
14:                 remove last triplet from $L_t$
15:             end if
16:         end for
17:     end for
18: end for
```

Algorithm 5 dfa_carving_backward_prop(A, A_{sort}, ζ_L)

```
Input: $A = (Q, \Sigma, \delta, q_I, F)$, carving-prep. DFA
       $A_{sort} : Q^{|Q|-1}$, topological sort of $A$
       $\zeta_L$, map of states to top backwards trans.
Output: $A$ with states to keep marked
        $\zeta_{TR}$, map of transitions $(s_s, \sigma, s_t)$ in $A$ to sets of ranks in $\mathcal{P}(N)$
 1: $s_f \leftarrow$ last state in $A_{sort}$
 2: $k' \leftarrow |\zeta_L(s_f)|$          ▷ number of top paths found
 3: for each $i = 1 \ldots k'$ do          ▷ init. $s_f$ rank sets
 4:     $\zeta_{SR}(s_f)[i] \leftarrow \{i\}$
 5: end for
 6: for each $s_t \in$ reverse($A_{sort}$) except last do
 7:     $SR_t \leftarrow \zeta_{SR}(s_t)$
 8:     if $SR_t \neq \perp$ then          ▷ no ranks for $s_t$
 9:         continue          ▷ skip $s_t$ rank propagation
10:     end if
11:     mark $s_t$          ▷ $s_t$ is to be kept
12:     $L \leftarrow \zeta_L(s_t)$
13:     init. $\zeta_{BR}$ as an empty map of $s_t$ backwards transitions in $(\Sigma, Q)$ to lists of rank sets
14:     for each $i \in 1 \ldots |SR_t|$ do
15:         $(w, \sigma, s_s) \leftarrow L[i]$
16:         if $\zeta_{BR}(\sigma, s_s) = \perp$ then
17:             $\zeta_{BR}(\sigma, s_s) \leftarrow$ empty list
18:         end if
19:         append $SR_t[i]$ to $\zeta_{BR}(\sigma, s_s)$
20:     end for
21:     for each $(\sigma, s_s, BR) : \zeta_{BR}(\sigma, s_s) = BR$ do
22:         $\zeta_{TR}(s_s, \sigma, s_t) \leftarrow \bigcup_{R \in BR} R$
23:         if $\zeta_{SR}(s_s) = \perp$ then
24:             $\zeta_{SR}(s_s) = \emptyset$
25:         end if
26:         $SR_s \leftarrow \zeta_{SR}(s_s)$
27:         for each $i = 1 \ldots |SR_t| - |SR_s|)$ do
28:             append $\emptyset$ to $SR_s$
29:         end for
30:         for each $i = 1 \ldots |SR_s|$ do
31:             $SR_s[i] = SR_s[i] \cup SR_t[i]$
32:         end for
33:     end for
34: end for
35: mark $q_I$
```

Algorithm 6 dfa_carving_cleanup(A, A_{sort}, ζ_L, ζ_{TR})

```
Input: $A = (Q, \Sigma, \delta, q_I, F)$, a DFA that underwent carving backward propagation
       $A_{sort} : Q^{|Q|-1}$, topological sort of $A$
       $\zeta_L$, map of states to top back. transitions
       $\zeta_{TR}$, map of transitions to rank sets
Output: $A$ after clean up
 1: $s_f \leftarrow$ last state in $A_{sort}$
 2: for each $(w, \sigma, s_s) \in \zeta_L(s_f)$ do ▷ note $\sigma = \varepsilon$
 3:     remove transition $\delta(s_s, \sigma) = s_f$
 4:     $F \leftarrow F \cup \{s_s\}$
 5: end for
 6: for each $s_s \in A_{sort}$ do
 7:     if $s_s$ is marked then
 8:         for each $(\sigma, s_t) : \delta(s_s, \sigma) = s_t$ and $\zeta_{TR}(s_s, \sigma, s_t) = \perp$ do
 9:             remove transition $\delta(s_s, \sigma) = s_t$
10:         end for
11:     else remove $s_s$ from $A$ along with all transitions from $s_s$
12:     end if
13: end for
14: remove $s_f$ from $A$
```

B Supplementary Material

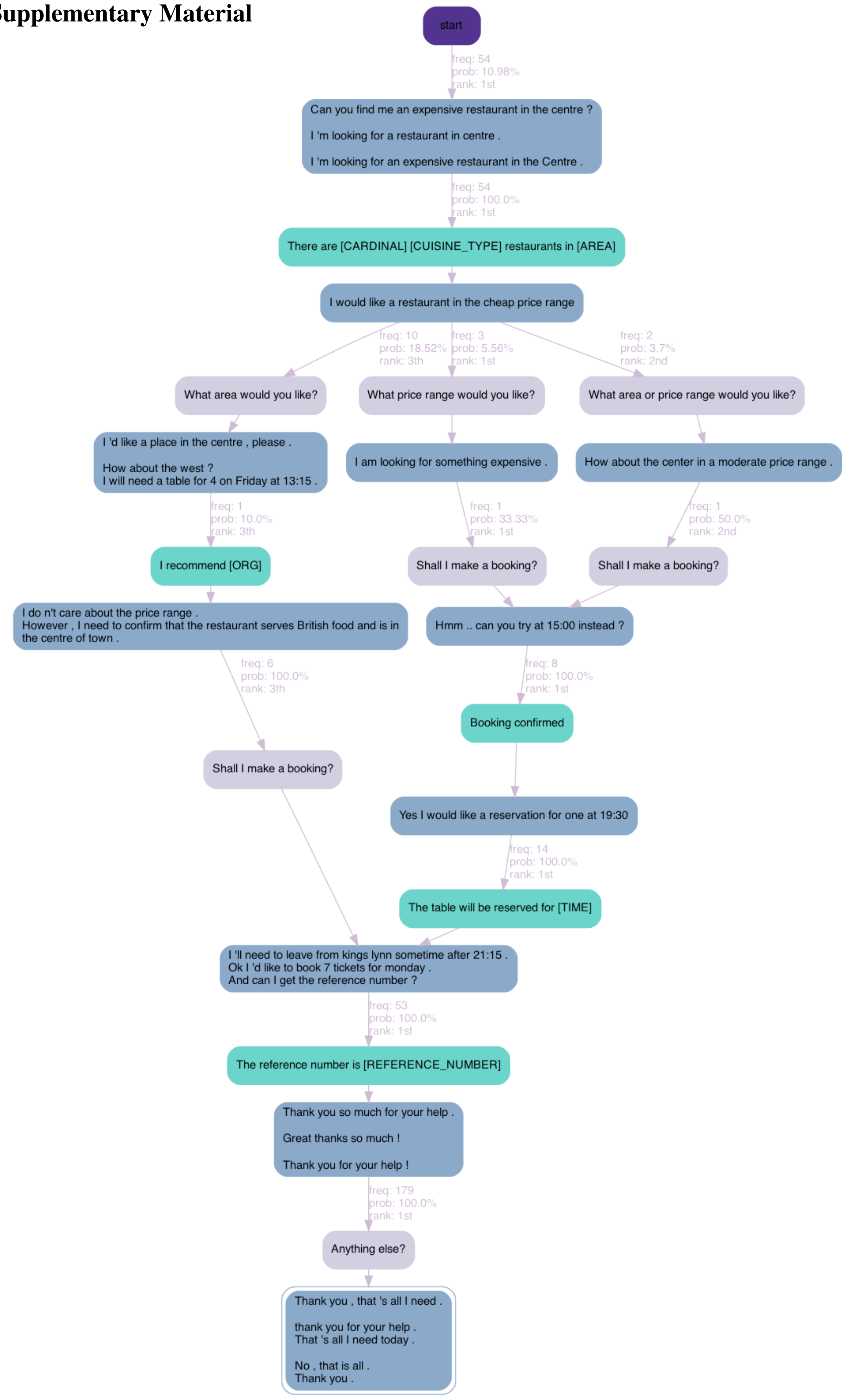

start
freq: 54
prob: 10.98%
rank: 1st
Can you find me an expensive restaurant in the centre ?
I 'm looking for a restaurant in centre .
I 'm looking for an expensive restaurant in the Centre .
freq: 54
prob: 100.0%
rank: 1st
There are [CARDINAL] [CUISINE_TYPE] restaurants in [AREA]
I would like a restaurant in the cheap price range
freq: 10
prob: 18.52%
rank: 3th
freq: 3
prob: 5.56%
rank: 1st
freq: 2
prob: 3.7%
rank: 2nd
What area would you like?
What price range would you like?
What area or price range would you like?
I 'd like a place in the centre , please .
How about the west ?
I will need a table for 4 on Friday at 13:15 .
I am looking for something expensive .
How about the center in a moderate price range .
freq: 1
prob: 10.0%
rank: 3th
freq: 1
prob: 33.33%
rank: 1st
freq: 1
prob: 50.0%
rank: 2nd
I recommend [ORG]
Shall I make a booking?
Shall I make a booking?
I do n't care about the price range .
However , I need to confirm that the restaurant serves British food and is in the centre of town .
Hmm .. can you try at 15:00 instead ?
freq: 6
prob: 100.0%
rank: 3th
freq: 8
prob: 100.0%
rank: 1st
Booking confirmed
Shall I make a booking?
Yes I would like a reservation for one at 19:30
freq: 14
prob: 100.0%
rank: 1st
The table will be reserved for [TIME]
I 'll need to leave from kings lynn sometime after 21:15 .
Ok I 'd like to book 7 tickets for monday .
And can I get the reference number ?
freq: 53
prob: 100.0%
rank: 1st
The reference number is [REFERENCE_NUMBER]
Thank you so much for your help .
Great thanks so much !
Thank you for your help !
freq: 179
prob: 100.0%
rank: 1st
Anything else?
Thank you , that 's all I need .
thank you for your help .
That 's all I need today .
No , that is all .
Thank you .

Figure 12: Dialog flow for top 3 restaurant booking paths. Bubble colors are: purple for the dialog start (initial state), blue for customer utterances (DFA states), and gray/green for agent questions/responses (DFA transitions).

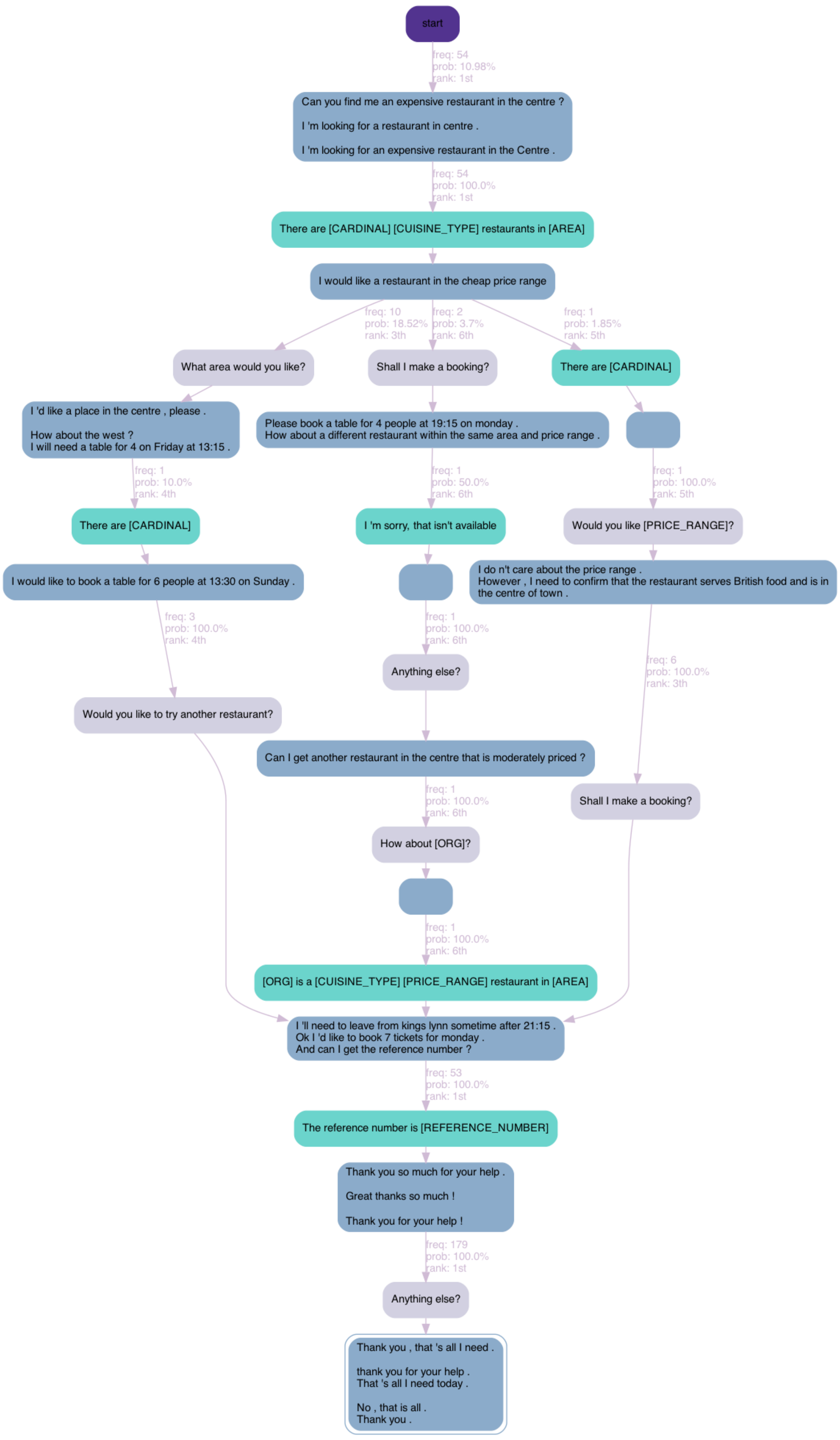

start
freq: 54
prob: 10.98%
rank: 1st
Can you find me an expensive restaurant in the centre ?
I 'm looking for a restaurant in centre .
I 'm looking for an expensive restaurant in the Centre .
freq: 54
prob: 100.0%
rank: 1st
There are [CARDINAL] [CUISINE_TYPE] restaurants in [AREA]
I would like a restaurant in the cheap price range
freq: 10
prob: 18.52%
rank: 3th
freq: 2
prob: 3.7%
rank: 6th
freq: 1
prob: 1.85%
rank: 5th
What area would you like?
Shall I make a booking?
There are [CARDINAL]
I 'd like a place in the centre , please .
How about the west ?
I will need a table for 4 on Friday at 13:15 .
Please book a table for 4 people at 19:15 on monday .
How about a different restaurant within the same area and price range .
freq: 1
prob: 10.0%
rank: 4th
freq: 1
prob: 50.0%
rank: 6th
freq: 1
prob: 100.0%
rank: 5th
There are [CARDINAL]
I 'm sorry, that isn't available
Would you like [PRICE_RANGE]?
I would like to book a table for 6 people at 13:30 on Sunday .
I do n't care about the price range .
However , I need to confirm that the restaurant serves British food and is in the centre of town .
freq: 3
prob: 100.0%
rank: 4th
freq: 1
prob: 100.0%
rank: 6th
freq: 6
prob: 100.0%
rank: 3th
Anything else?
Would you like to try another restaurant?
Can I get another restaurant in the centre that is moderately priced ?
Shall I make a booking?
freq: 1
prob: 100.0%
rank: 6th
How about [ORG]?
freq: 1
prob: 100.0%
rank: 6th
[ORG] is a [CUISINE_TYPE] [PRICE_RANGE] restaurant in [AREA]
I 'll need to leave from kings lynn sometime after 21:15 .
Ok I 'd like to book 7 tickets for monday .
And can I get the reference number ?
freq: 53
prob: 100.0%
rank: 1st
The reference number is [REFERENCE_NUMBER]
Thank you so much for your help .
Great thanks so much !
Thank you for your help !
freq: 179
prob: 100.0%
rank: 1st
Anything else?
Thank you , that 's all I need .
thank you for your help .
That 's all I need today .
No , that is all .
Thank you .

Figure 13: Dialog flow for top restaurant booking paths 4 to 6. Bubble colors are: purple for the dialog start (initial state), blue for customer utterances (DFA states), and gray/green for agent questions/responses (DFA transitions).