Model Inference for Multi-turn Q&A Dialogues

Xiaoqiang Yan, Ge Shi, Chen Qu, Zhaoduo Wen, Hao Liu
College of Information and Computer Sciences, University of Massachusetts Amherst
{xyan, geshi, chenqu, wenzhaoduo, haoliu}@cs.umass.edu

Abstract

Model inference is to produce a model that describes every details of a process, an execution, a procedure, or any event of the like, by analyzing traces(logs, records, observations,etc.) of that given event. Synoptic and InvariMint are two power tools for model inference. Our project aims to assess the inferred models from both software in terms of accuracy, expressiveness and conciseness. We generated a large number of traces from multi-turn question and answer conversations. We conducted extensive experiments. We performed both qualitative and quantitative analysis of the inferred models. We managed to make some insightful observations. Our research shows that, given a large number of traces, model inference can tackle the complexity of human conversations, and provide insights on several patterns of Q&A dialogues.

1. Introduction

Model inference uses a set of observations of how a process executes to the produce a model of everything the process can do. A typical application is to execute a software system many times and record logs of their executions. These logs (traces) could be the methods that the system executes, or the logging information developers choose to put into the system. A model inference tool takes these log traces to produces a model of all possible system behaviors, both observed and unobserved.

This paper presents the experiment using Synoptic, a software tool that helps developers to infer a concise and accurate system model, to construct models for multi-turn QA conversations. The tool is intended to help programmers debug and attain insights into how a system runs. Synoptic processes the logs of a system and generates a model similar to a finite state machine. The model is subject to the temporal invariants mined from the logs. By examming existing logs, Synoptic produces detailed models which could be conducive for developers to understand the system in question. In this paper, we will not dive into the algorithm and implementation details of Synoptic the software. Nevertheless, it is necessary to mention some of the terminologies used in Synoptic software.

Synoptic takes a log file that contains execution traces and a set of user-defined regular expressions as inputs.The regular expressions are used to parse the log file and extract an event instance from the log lines. A trace is a linear graph built from the logs using the provided regular expressions each vertex is an event instance, and the directed edges represent the total ordering. From the trace graph, Synoptic mines three kinds of temporal invariants: (1) Whenever the event type a appears, the event type b always appears later in the same trace; (2) Whenever the event type a appears, the event type b never appears later in the same trace; (3) Whenever the event type b appears, the event type a always appears before b in the same trace. the term ”invariants” means the temporal event type relationships that must hold true over all the input traces.

Then, Synoptic generates a initial model which is a partition graph of the trace graph. In the Initial model, there is one partition per event type containing all the event instances of the type. This model is also generative it may accept traces that were not present in the log. Coarsening and refinement are dual operations on a Synoptic model. Synoptic refines partitions by splitting the graph until it satisfies all the mined invariants. The algorithm may end up refining more than it needs to. When this happens, the model will contain partitions that can be merged without violating the satisfied invariants. Here involves the final coarsening to merge such partitions.

The whole algorithm flow is shown in Figure 1. The corresponding time cost for extracting trace graph, mining trace graph, partition, refinement and coarsening are parsing time, mining time, creating Graph time, splitting time and merging time. These will be important criterion to evaluate the model inference process.

2. Traces Generation

Inspired by the countless online QA (Question & Answering) forums such as Stack Exchange and Yahoo Answers!, we observe that there might be flow patterns in multi-turn QA conversations, and model inference can be a

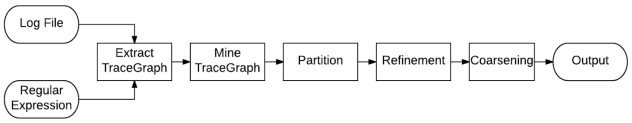

Figure 1. Introduction to the Inference Algorithm

powerful technique for mining the patterns. In this section, we describe the three stages of generating traces of multi-turn QA dialogues, which are dialogues collection, intent annotation, and traces extraction.

2.1. Dialogues Collection

We scraped thousands of dialogues from Microsoft Community, an online forum that provides technical support for a wide range of Microsoft products. All the dialogues have multiple turns and at least two participants. To ensure the quality and consistency of the dialogues, we selected over 2,000 dialogues with reasonable length for the annotation.

2.2. Intent Annotation

We used Amazon Mechanical Turk, an online crowdsourcing platform to annotate each utterance of the dialogues with user intent. We classify user intents in the dialogues into 12 classes as an extension of the classes defined in [2] as in Table 1. Each utterance can be assigned multiple labels because the co-occurrences of intents can be quite often.

2.3. Traces Extraction

We convert the flow of intents in dialogs into traces that can be interpreted with Synoptic and other model inference tools. Because of the complexity and diversity of human conversations, each utterance are labeled with one or more intents. Our method is to disentangle the multiple labels by only using one label each time. For example, if we have the labeled dialog of "Turn 1: OQ, GG, Turn 2: PA, Turn 3: PF", we turn them into two separate traces, the first one is "Turn 1: OQ, Turn 2: PA, Turn 3: PF", and the second one is "Turn 1: GG, Turn 2: PA, Turn 3: PF". We believe the model inference tools can capture this separation and merge them in the model.

We also filter out the dialogs with too complicated intents in each utterance. After the filtering, we have 2,034 dialogs that generate 27,975 traces with 147,610 events for the experiments.

3. Experiments with Synoptic

In this section, we present experiments with Synoptic. The traces generated in Section 2 are fed into Synoptic, which efficiently produces models that are consistent with our traces. Each node in the model represents an event type (user intent) with the edges weighted by transitions probability. First, we conducted simple experiments with traces generated from a single dialogue to get familiar with the tool and followed by vast amount of traces to generate general flow patterns.

In single dialogue experiments, we find some novel ways to interpret the models, which is to evaluate the quality of the conversation in terms of whether the users get the answers they want and to diagnose annotation errors of the labels. In vast traces experiment, we manage to get general flow patterns of multi-turn QA conversations and gain insights by making observations of the generated model. We also conducted systematic analysis of the sensitivity and generalization ability of models with respect to different number of traces.

3.1. Interpretation of Model from a Single Dialogue

3.1.1 Evaluation of Conversation Quality

Figure 2 shows the model generated using the 16th trace. In this trace, the user gives an original question and the agent provides a correct answer. The user then gives positive feedback and gratitude to the agent.

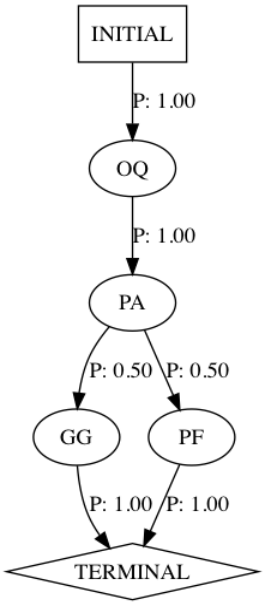

Figure 2. Conversation with Good Quality

The model shows that the initial state is always OQ, which is always followed by PA, transitioning with half chance to GG or PF. Both of them always go to the terminal state. This is the consistent with the input trace file. Therefore, we draw the conclusion that the quality of this single dialog is good from the model visually.

We also generate another model for comparison, which is shown in Figure 3. In this model, NF goes to terminal

Table 1. Description of Classes

Code	Label	Description
OQ	Original Question	The first question that initiates the QA conversation.
RQ	Repeat Question	Posters other than the user repeat a previous question.
CQ	Clarifying Question	User or agent ask for clarifications.
FD	Further Details	User or agent provides more details.
FQ	Follow Up Question	User asks for a follow up question about a relevant issue.
IR	Information Request	Agent ask for information of users.
PA	Potential Answer	A potential answer or solution provided by agents.
PF	Positive Feedback	User gives a positive feedback if the solution worked.
NF	Negative Feedback	User gives a negative feedback if the solution did not work.
GG	Greetings / Gratitude	Greetings or expressing gratitude.
JK	Junk	There is no useful information in the post.
O	Others	Posts that cannot be categorized using above guidelines.

state directly, which means the user doesn't get any reply after his/her negative feedback. This indicates that the quality of the conversation is not ideal.

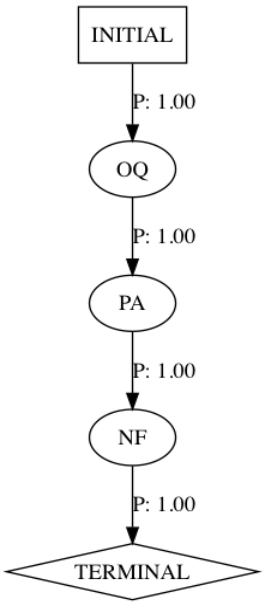

Figure 3. Conversation with Poor Quality

3.1.2 Locating Annotation Errors

The annotation of traces is conducted by crowdworkers, so errors are inevitable. Synoptic models capture event type orders and co-occurrence frequencies among event types. We could locate the possible annotation errors through Synoptic-generated models visually, which resembles the original purpose of Synoptic to diagnose malfunctioning software.

For example, model of the 40th log is shown in Figure 4. In this model, we can see after initial state OQ, the agents may give PA or GG both with 50% chance. PA, RQ or NF will follow GG all with 33% chance if the agent provides GG. Intuitively, the probability of GG immediately followed by NF without PA should be relatively low. Hence, we check the original 40th dialogue and find that the annotation has some errors.

3.2. Generating Flow Patterns of QA Dialogues

Observing models generated by a single trace is not sufficient to conclude the flow pattern in QA conversation. Thus, we input more traces to get a better and more general model.

We perform a systematic analysis on how the number of traces influence the model inference in terms of time spent on various stages, number of nodes, and number of invariants mined. The results are presented in Table 2 and Table 3.

Table 3. The Variants of Running Synoptic

Mined Invariants	Nodes	Sum time
62	45	9635
33	65	51350
3	29	71974
0	14	13251
0	14	37742
0	45	38580

From different models and above tables, we have several obervations in terms of time spent, number of invariants mined, and number of nodes. From the 1st to 3rd line of entries of Table 2, the time of generating models and the number of traces are linearly correlated. When the number of traces increases to 1000, the splitting and merging time drop significantly due to the mined invariants decrease to zero (Table 3). Given the complexity and diversity of human conversations, it is not surprising that the designed invariants in [1] may not exist in a large amount of traces. Thus, with no criterion to satisfy during partitioning and

Table 2. Time of Running Synoptic to Generate Models of Vast Traces (ms)

Dialogue #	Traces #	Parsing Time	Generate Relation Time	Mining Time	Create Graph Time	Splitting Time	Merging Time	Exporting Time	Total Time
20	390	67	67	63	6	6450	1422	1560	9635
100	1324	127	281	125	12	36096	6907	7802	51350
500	7151	425	2893	329	51	57970	8061	2245	71974
1000	14521	752	10177	658	80	**5**	**2**	1577	**13251**
2000	27511	1945	32422	1078	157	5	2	2133	37742
2034	27975	1849	33693	815	253	6	3	1961	38580

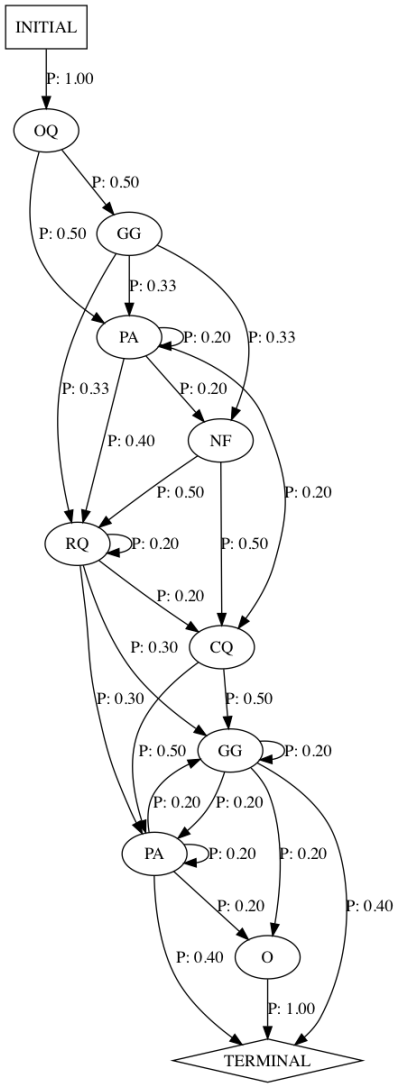

Figure 4. Model for Locating Annotation Errors

merging, the generated graph can be very concise with only 14 nodes, which are the 12 labels with initial and terminal states.

The Splitting time accounts for most part of the total time of running Synoptic to generate models as shown in Table 2. To verify that the splitting time plummets at the time that the number of invariants drops from one to zero rather than decreases gradually. We experiment on the scenarios of the number of invariants from four to one and inspects the splitting time and number of nodes. In Tabel 4, we notice that the splitting times are more than 40000 milliseconds when the number of mined invariants is positive. However, when the number of invariants become zero, the time cost for splitting is just 5 milliseconds. Even with equal number of invariants, the number of nodes and splitting time are not stable.

Table 4. The Splitting Time and Number of Nodes

Lines	Mined Invariants	Nodes	splitting time
40000	4	32	42916
45000	4	32	56509
48000	4	38	56966
49000	4	32	75222
49120	3	33	61314
50000	2	34	44179
70000	1	24	44815
100000	0	14	5

3.3. Generalization Analysis

Model inference is a generative process, so it is important to test the generalization ability of the model. Since we are investigating how the number of traces influence the generalization ability of the model, we believe it is unfair to use a fixed test set (with fixed number of testing traces). So we build upon the different number of traces used in the last section, and split them respectively into train/test sets with 10% of the traces as test set, making sure the size of the test sets are proportional to the training sets that are used to infer the models. The metric is the generalization ability, which is the percentage of testing traces that exist in the model.

As presented in Table 5, generalization ability has a major improvement from 20 traces to 100 traces, and does not change much in general since then. The best performance is at 500 traces. Due to the complexity of human QA conversations, the generalization ability is not so ideal. We believe there might be a another major improvement if the number

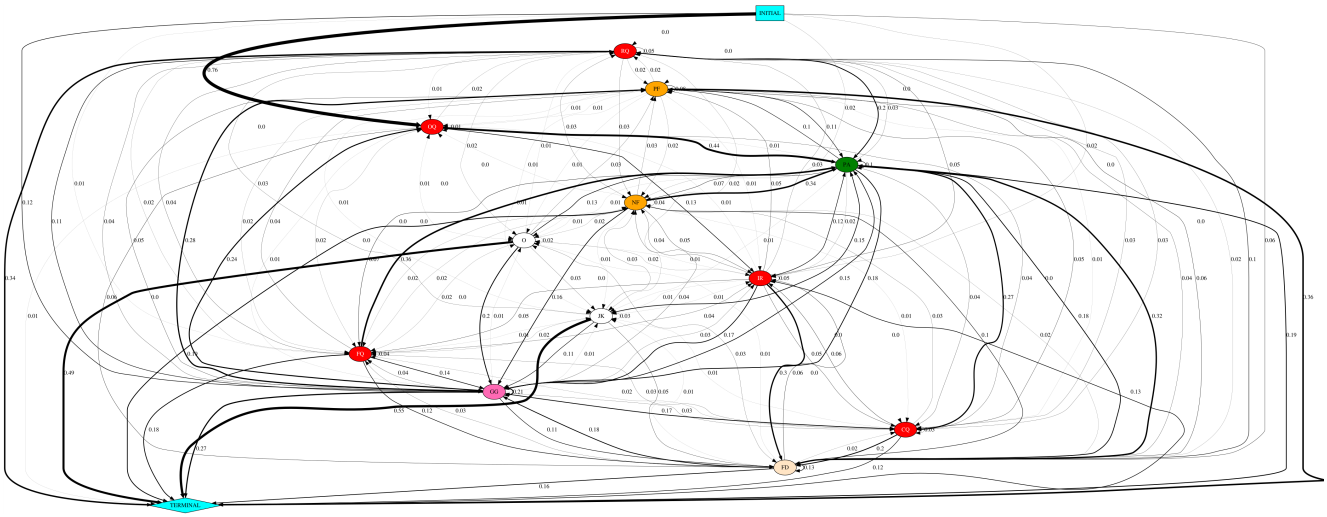

Figure 5. Model Visualization

of training traces is a magnitude larger. Given the limited resources on annotation, we are not able to conduct further experiments for now.

Table 5. Generalization Ability

Number of Traces	Generalization Ability
20	38.46%
100	63.16%
500	**63.27%**
1000	62.17%
2000	62.97%
2034	62.79%

4. Observations

4.1. Model Visualization

We processes the dot file generated by the model inference tool to add some styles to facilitate the analysis as in Figure 5. All edges are weighted by transition probability and nodes are marked with different colors. Labels that can be categorized as a question are marked red, both feedbacks are marked orange, and "initial", "terminal" nodes are marked blue.

4.2. Model Refinement

Based on Section 3.1.2, we find some errors in the annotation. One of our primary objectives in this section is to refine the annotation to eliminate errors. After scrutinizing the model, we find some potential false annotations, such as GG immediately followed by NF. This kind of unintuitive transitions are typically with less than 5% probabilities in the model generated from all traces. In order to keep cardinal edges only and exclude all potential false annotations, the edges of less than 5% probabilities should be removed. Figure 6 shows the result of refinement which is much more concise than before.

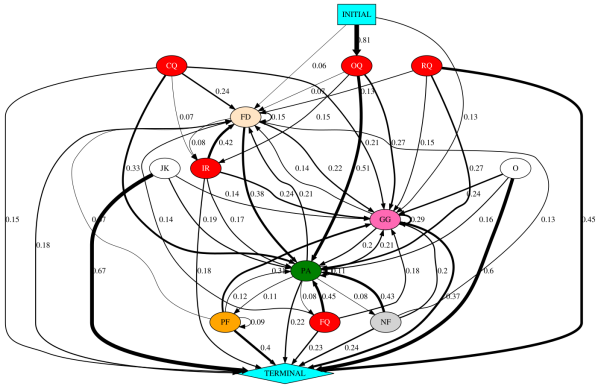

Figure 6. Model Visualization

The result of model refinement via increasing traces is evaluated by the ratio of the number of deleted edges to the number of all edges (RD/A) and the corresponding mean probability (MDP) for each node in the graph. From Table 6, we observe that with the number of traces increasing, deeper refinement can be made to the model reflected by higher RD/A and MDP. Thus, the model reaches a relatively stable state in the end. This also means fewer transition errors caused by potentially false annotations in the model with more traces, for there are few of them contributing to more than 5% probabilities. Thus, we can conclude that increasing traces improves the stability and accuracy of the model.

Table 6. Refinement Results

Number of Traces	RD/A	MDP
20	29.5% (62/210)	3.24%
100	45.7% (192/420)	6.38%
500	55.1% (119/216)	9.96%
1000	64.4% (105/163)	15.0%
2000	67.8% (114/168)	17.0%
2034	67.2% (113/168)	16.6%

4.3. Quantity Properties

In terms of a typical Q&A process, we want to get insight into how many steps from Initial to Terminal make an typical conversation. One transition from a state to another in the finite state machine denotes a step. This can be done by counting the number of steps from Initial to Terminal when walking the graph according to transition probability. The experiment result is based on the walking of the refined model for 10,000 times. The mean of numbers of steps to finish a conversation is 4.56 and the median is 4 as are shown in Figure 7 .

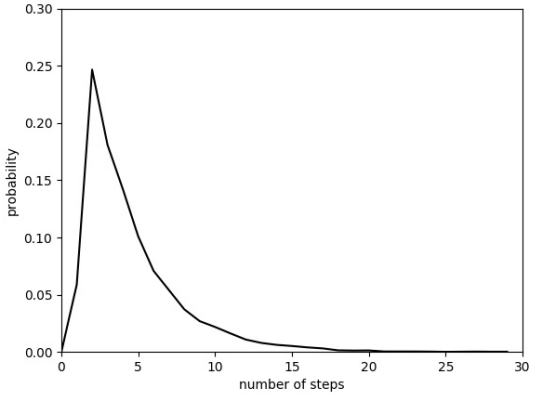

Figure 7. Estimate Steps

4.4. Model Observations

Some very intuitive observations can be made from Figure 5:

- Most dialogues begin with an ”original question”, sometimes accompanied with ”greetings/gratitude”.
- ”Original question” tends to lead to ”potential answer”.
- ”Information request” and ”clarifying question” tend to lead to ”further details”.
- ”Positive feedback” tends to lead to the end of the dialog. while ”negative feedback” tends to lead to potential answers.
- Dialogues tend to end after ”others” or ”junk”.
- More traces increases the stability and accuracy of model.
- The mean and median of numbers of steps to complete a conversation are 4.56 and 4.

All the observations could be useful when designing a dialogue system.

5. Synoptic vs. Invarimint

We conduct experiments with the same traces using InvariMint. We found the model generated from two models are consistent in terms of simple traces as presented in Figure 8 and may differ visually from each other as shown in Figure 9 and 10, which are generated from the same trace.

Synoptic models use nodes to represent states, while InvariMint models use edges. For models with complicated relations among states, Synoptic models are easier to interpret.

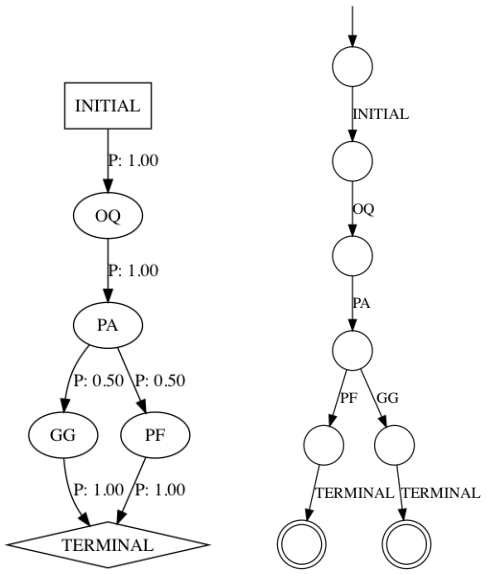

Figure 8. Consistent Models

Table 7 is the statistics comparisons between Synoptic and Invarimint on the small traces regarding to total number of traces. Note that The algorithm used in Invarimint is “InvariMint-Synoptic”. The experiment is run on a Macbook Pro 2016, with the processor being 2.6GHz, Intel Core i7, memory being 16 GB 2133 MHz LPDDR3 and disk being 256GB SSD.

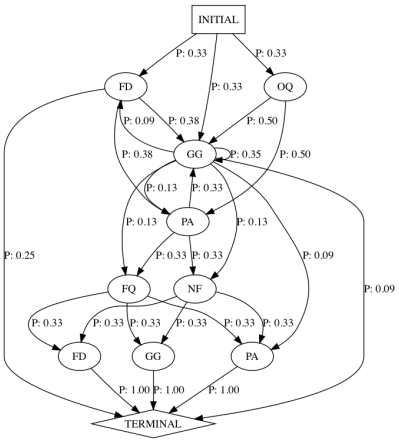

Figure 9. Synoptic Model

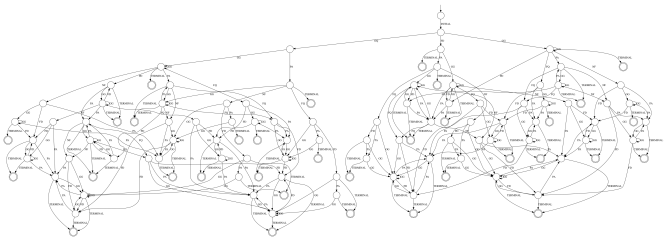

Figure 10. InvariMint Model

Table 7. Comparison of Running Synoptic vs InvariMint. Parsing and mining time are in millisecond while running time is in second

Traces #	Parse Time	Mining Time	Mined Invariants	Nodes #	Run Time
2	7/4	6/4	19/19	6/8	0.09/1.25
2	9/4	5/4	46/46	8/10	0.49/1.93
72	24/48	23/15	25/25	11/109	1.13/2.41
12	13/8	10/11	36/36	10/15	0.64/1.76
26	11/16	11/12	26/26	11/32	0.26/1.85
54	15/41	16/10	10/10	11/118	1.22/2.71

In terms of total number of traces and mined variants which are only related to input data, Invarimint and Synoptic have same values. However, the model generated by Invarimint has much more nodes than that of Synoptic. One obvious reason is that Invarimint doesn't merge terminal node as Synoptic does. We should also note that the model generated by Invarimint is not same as Synoptic with the increasing of total number of traces as shown in Figure x, which makes the model generated by Invarimint be much more complicated than that of Synoptic, and thus having more nodes.

We measure the running time by terminal command "time". Note that Invarimint usually perform better on mining. However, we find that Invarimint has a larger running time due to two reasons. First, Invarimint outputs more logs to the terminal. Second, Invarimint needs much more time to export model graph as its generated models are much more complicated.

To sum up, we can see that the models generated by Invarimint are significantly different from models generated by Synoptic with regard to nodes of models. The models generated by Synoptic are much more clear, which will help a lot on analyzing. In addition, InvariMint generally has a larger running time due to two reasons while needs less time to mine the traces.

6. Conclusion

In our project, we explored model inferrence, its theory and its practice. We use Synoptic the software to investigate and produce a model for multi-turn QA conversations. The model accurately captures the important characteristics and certain patterns of human question answering dialogues. We conducted extensive experiments and analyzed both qualitatively and quantitatively to examine the model inference process in trying to attain a deeper and more insightful understanding of the subject.

Based on our research and experiments, we draw the following conclusions:

1. Increasing traces could give rise to more accurate and stable models. We evaluate the model refinement via RD/A and MDP as described in Section 4.2 and we find that simple traces introduce bias upon generating a model due to false annotations. In addition, models become increasingly stable with the increasing of the number of traces. As listed in Table 6, the last three models have almost the same RD/A and MDP. However, increased number of traces contributes little, if any, to compose a more accurate model beyond certain threshold. Hence, it's highly recommended to fine-tune the number of traces for optimal models.

2. Splitting time can be significantly affected by the number of invariants. As shown in Table 4, splitting time suddenly drops to 5ms when mined invariants reaches 0.

3. We walked the generated graph according to transition probability at every step and find that a QA dialogue typically contains 4.56 turns as in Figure 7.

4. Synoptic and InvariMint comparison. InvariMint performs better in terms of mining time while Synoptic gives a more human readable representation of the model and the time to generate the model graph is significantly less. According the the paper, "Using Declarative Specification to Improve the Understanding, Extensibility, and Comparison of Model-Inference Algorithms"[1], InvariMint is capable of using declared invariants and thusly more expressive. However in this experiment, we don not have a prior knowledge about model therefore we are unable to declare any invariants.

References

[1] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *ESEC/FSE 2011: The 8th joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 267–277, Szeged, Hungary, Sept. 2011. 3

[2] S. Bhatia, P. Biyani, and P. Mitra. Classifying user messages for managing web forum data. In *Fifteenth International Workshop on the Web and Databases (WebDB 2012)*. Citeseer, 2012. 2