Dan Kondratyuk

April 30, 2017

Improving the Memory of Intelligent Personal Assistants

A man doesn't know what he knows until he knows what he doesn't know.

- Lawrence J. Peter

1 Introduction to Intelligent Personal Assistants

Intelligent personal assistants (IPAs) of today—e.g., Google's Assistant, Apple's Siri, and Microsoft's Cortana—are rapidly transforming how humans interact with machines. These software systems aim to provide a natural interface between humans and machines using speech and text. Such a form of communication with tech allows for users to execute quick and simple tasks with just a few movements of the jaw, or a few taps of the thumbs. However, the technology is still in its early stages. The following dialogue taken from a conversation with Apple's Siri assistant illustrates this point (Reddit 2016).

USER: Hey Siri, remind me this weekend to install the car seat.

SIRI: OK, I'll remind you to 'Install the car seat'. When would you like to be reminded?

USER: Saturday.

SIRI: Ok, just tell me what you want to be reminded about.

USER: What?

SIRI: OK, I'll remind you about 'What' on Saturday at 9:00 AM.

IPAs commonly behave like amnesiacs; they often do not remember what the user told them seconds ago, and are deficient in recognizing the content of their own messages. Although these assistants are capable of performing complex tasks, with so little shared context the users are left repeating themselves each time they want the system to perform one of these tasks. When a user refers to a concept by name (e.g., ‘George’), the IPA does nothing to better understand its underlying meaning. This inability to recall severely inhibits the IPA’s ability maintain natural conversation.

If these obstacles could be overcome, communication with IPAs would lead to far more fluid, human-like interaction, thereby increasing their usefulness. Humans can rapidly adapt to new situations by defining concepts and forming relationships mid-conversation, and an intelligent software agent should be no different. Users prefer to communicate with something that can *share* these concepts with natural language.

In this document, I introduce and evaluate a simple memory model that allows the IPA’s various components to leverage high-level concepts extracted from its past dialogue history, called the **dynamic memory IPA** model (DyMIPA). It is designed to be *flexible*, incrementally introducing new types of concepts when needed—and *expandable*, able to be improved and upgraded without a major redesign. When incorporating these techniques into intelligent personal assistants, the assistant can drastically improve its appearance of natural interaction.

This paper is split up into five sections. This introductory section provides preliminary information about IPA motivation and design. The second section explains the major design decisions behind constructing the dynamic memory model. The third section defines the necessary components to this model. The fourth section highlights a software implementation which evaluates answers to a simple problem domain, and the fifth section summarizes results and indicates future work.

1.1 IPA Motivation

Within the last five years, demand for increasingly capable IPAs has intensified, and competition has been nothing less than fierce. The most prominent corporations in the tech industry, Google, Apple, Microsoft, and Amazon, have developed and released versions of their own intelligent assistants, which are all vying to dominate the AI spotlight. This accelerated growth has also sparked several research efforts. In early 2016, IBM launched their IBM Watson AI Xprize initiative, offering $5 million in prizes to competing teams for the most groundbreaking and innovative AI approaches from the first round in 2017 to the final round in 2020 (IBM 2016). A few months later in September, Amazon unveiled the Alexa Prize, another AI competition to award the teams with the best conversational bots $2.5 million in prizes in November 2017 (Amazon 2016). And just a month after that, a team of Microsoft researchers announced that they could reach human parity in conversational speech recognition, otherwise known as speech to text (Linn 2016).

One may argue that it is most reasonable to create an IPA user interface that users can click on, gesture to, and type in without the need to resolve vague and ambiguous expressions that natural language introduces. Although reasonable, it may not be the best option available. Lison (2014, p. 1), argues that,[1]

> Spoken language is one of the most powerful systems of communication at our disposal. A large part of our waking hours is spent in social interactions mediated through natural language. The central role of spoken language in our daily lives is largely due to its remarkable ability to convey (sometimes highly elaborate) ideas in a robust and efficient manner.

Natural language, both written and spoken, allows for expressiveness unmatched by more simple and structured means of interaction. IPAs benefit from this by using language as a means to blend many disparate tasks together, blurring the lines between the modules necessary to perform each task. In fact, spoken language in particular is one of the simplest and easiest ways a human can communicate, as “face-to-face conversation is the basic and primary use of language . . . the principle setting which doesn’t require any social skills” (Kennington 2016, p. 20), see also (Fillmore 1981, p. 152). Thus, ease of use and expressiveness are two compelling reasons why natural language is a suitable choice for assisting users intelligently.

To create a system that can both comprehend and produce human language is is no small

[1]See also (H. H. Clark 1996)

undertaking, but the usefulness of these models cannot be understated, as they are used *everywhere* in the modern computing realm. Take, for example, how Google takes advantage of the information learned from building their IPA systems. Google integrates several natural language understanding approaches to enhance web search relevance, neural networks to reduce the cost of cooling their data centers, natural language generation functions to translate one language into another, machine learning algorithms for their predictive analytics platform, and all of these techniques to some extent have been introduced in their new IPA, Google Assistant (J. Clark 2016; Pichai 2016). It follows that if an IPA's effectiveness could be improved, the knowledge gained is valuable towards a symposium of endeavors.

1.2 IPA Architecture

IPAs come in many flavors. Some IPAs interact with users through voice to execute general tasks (e.g., Amazon Alexa can play music, set reminders, and tell jokes). Other IPAs can be constrained to text input, and focus its knowledge on a restricted domain, such as a pizza-ordering chatbot that asks a sequence of basic questions before submitting the order. But what unites all IPAs is their ability to process natural language input, decide what to do next based on that input, and execute some tasks resulting from that decision: whether it be responding back with natural language, displaying a formatted result, or submitting a request to an external service. This description is what is referred to as a dialogue system.

Dialogue systems form an integral component of IPA design, which are responsible for maintaining conversation with a user (Martin and Jurafsky 2000). They consist of the following three main components: (1) the **natural language understanding unit** (NLU), (2) **dialogue management unit** (DM), and (3) **natural language generation unit** (NLG). An example of a user interacting with the dialogue system illustrated in figure 1 (One can think of an IPA as a type of dialogue system which also contains external services and APIs it interacts with).

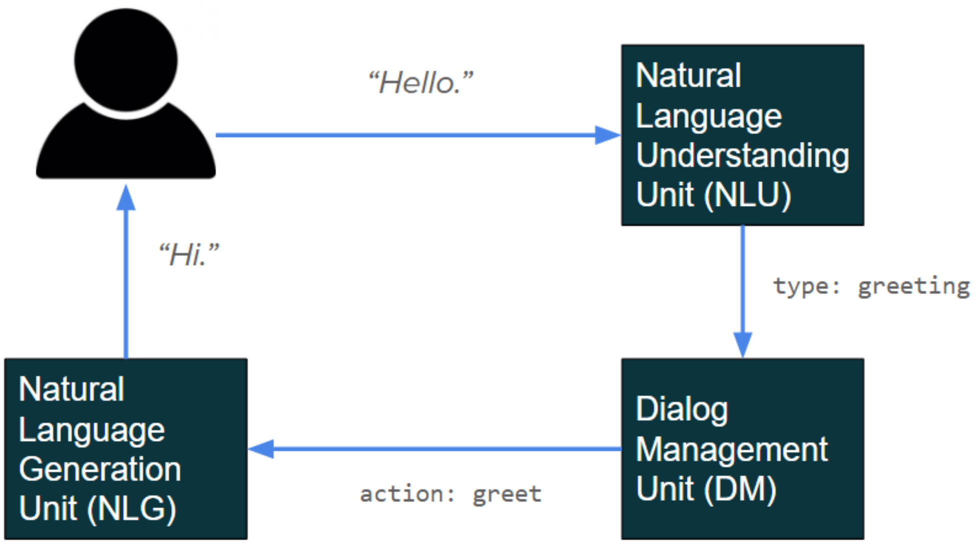

"Hello."
Natural Language Understanding Unit (NLU)
type: greeting
Dialog Management Unit (DM)
action: greet
Natural Language Generation Unit (NLG)
"Hi."

Figure 1: A Dialogue System Example

The dialogue system can receive natural language input in the form of text, and depending on its content, it can retrieve information or ask clarifying questions. All input from the user first enters the NLU, which parses incoming terms, extracts the relevant user intent into a machine-readable format.

Understanding, characterizing, and reasoning with sentences is a difficult problem due to complexities involving a person's underlying meaning and intent. Not only can language be ambiguous, it is highly dependent on the context of previous sentences, current interactions, and the situation of the dialogue. "Dialogue acts are strongly contextual in nature: their precise meaning can often only be comprehended within the particular conversational context in which they appear. The successful interpretation of dialogue acts must therefore venture beyond the boundaries of isolated utterances" (Lison 2014, p. 12). How the NLU handles this will affect how well the rest of the system will accurately respond to a request. This will become important in later sections when integrating the dynamic memory model.

Once the NLU has processed its input, it will produce a description of the user utterance. Depending on the information needed, this can include parts of speech (e.g., noun, verb, adjective), a collection of named entities like 'Martha' and 'New York', and probable action keywords, such as ‘book a flight’. In the example above, the NLU has parsed the user’s input of “Hello” to be an instance of an object of type `greeting`.

Upon producing relevant data about user input, the NLU can deliver its formatted information to the DM, which decides what to do with the information. The DM component interacts with the rest of the system using **Dialogue management**. “Common to virtually all approaches to dialogue management is (1) the representation of the system’s knowledge of the current situation in a data structure called the dialogue state and (2) the use of a decision mechanism to select the action to perform in each dialogue state” (Lison 2014, p. 23). Thus, given the current description of what the user has stated and all the descriptions of previous statements, the task of the DM is to determine what to do next.

Depending on what the NLU gave it, the DM can perform a variety of actions. If the DM infers that the user wishes to request a piece of information, the DM retrieves the corresponding information from an external service (e.g., a Google search) or an internal database (e.g., private user info). Alternatively, if the DM understands that the user wants to perform an action, the DM can notify the appropriate application to fulfill the request (e.g., setting a timer on the user’s smartphone). Figure 1 shows that the DM has decided to greet the user back with the `greet` action.

As a means of response, the DM can send signals to the NLG. The NLG receives this information and converts it into natural language that the user can read or hear. In short, the NLG is the reverse of the NLU: instead of converting natural language to structured descriptions, the NLG converts descriptions into language. Here, the NLU translated `greet` to the utterance “Hi.”

The user can then continue the conversation with the dialogue system, and the cycle repeats. More units and modules can be attached to the dialogue system to enhance its capabilities (e.g., automatic speech recognition), but the NLU, DM, and NLG are the most important for this discussion.

2 Design Decisions for Dynamic Memory

As mentioned in the introduction, one area where IPAs should improve is in their ability to form concepts in their memory and use them in their dialogue. This can be manifested in how well the system can extract relevant information from user utterances, store them for later use, and retrieve them when necessary. The major question that this paper will explore is ***how interaction could be stored in memory and how that information can be leveraged for future use***.

To understand how memory could play a role in dialogue, one must first understand the essence of *dialogue*.

> One of the most important properties of dialogue is that it is fundamentally a *collaborative activity*, with emphasis on both words. It is, first of all, an activity motivated by the desire to fulfill specific (practical or social) goals... Individuals participating in a dialogue routinely collaborate in order to coordinate their contributions and ensure mutual understanding, thereby making the interaction more efficient (Lison 2014, pp. 9-10).

IPAs have solid footing on the activity part of dialogue, as their entire purpose is to perform tasks for the user. The user would not initiate a dialogue if there was no worthwhile activity present with the IPA. The collaborative part needs improvement. There is some coordination and understanding between user and system, but a *mutual* understanding is elusive. The concepts present in the user's internal state needs to be reflected (to some degree) in the system.

The following section outlines some of the goals of the dynamic memory model so that it can take steps to increase mutual understanding.

2.1 The Goal of the Dynamic Memory IPA Model

Based on observations in commercially available IPAs, IPA design has been centered around the natural language understanding and dialogue management aspect of the dialogue system. For instance, Amazon's Alexa is marketed as a developer platform for skills and capabilities that the assistant can provide (*Amazon Developer* 2017), and Google Assistant offers similar features with their Actions SDK (*Actions on Google* 2017). This focus has greatly improved the depth and breadth of capabilities that IPAs can task, but one feature lacking from every new skill that they learn is in their understanding of the user. Effectively, these platforms are separate apps disguised as a single cohesive interface. Then it should be expected that each app is introduced with as little understanding of the user as every other app. These IPAs would benefit greatly if they could utilize some context of the user's preferences and knowledge domain.

To understand the dynamic memory IPA model effectively, one must keep into consideration its desired outcomes and goals (which respond to the deficiencies seen in current IPAs). A dynamic memory IPA should be:

- **Dynamic** - As information updates pour in, the system is able to accommodate data types and relationships that it has never encountered before.
- **Incremental** - The memory contained in the model can be updated incrementally. At no point will the core system need to be retrained or reevaluated when encountering new data.
- **Flexible** - The model can be upgraded and improved upon without a major redesign.
- **Interpretable** - The information contained within the model is easily observable, and it is possible to interface with a variety of services.

First and foremost, the model must exhibit *dynamic* memory. Human memory is highly malleable. It constantly changes and updates to handle the complex whirlwind of information encountered in day-to-day life. There is no rigid set of rules that govern what types of information are acceptable. This dynamic nature allows the meaning of new information to be learned rather than discarded. Likewise, the model should emulate this behavior. The repository of information available to the memory model should always be expanding and improving with each new utterance, rather than remaining as a static data store. Thus, it is necessary that the memory in the knowledge representation has as little prior knowledge and formal rules as possible to promote optimum flexibility.

Secondly, the model must be incremental in how it learns information. One such approach is to use an idea known as **incremental processing**.[2] Prominent IPAs use a *turn-based model*, in which dialogue is processed in chunks. This behavior closely mimics text messaging, in which the recipient (i.e., the IPA) receives messages in short bursts, incurring sizable delays in responsiveness. A potential problem with this model is that the IPA must wait for a message snippet to completed by the user before it can process the information. The *incremental processing model* aims to improve this by sending smaller units of data, namely letters or words, before the user even completes their statement. The dialogue system would then update their understanding of what the user intends to say incrementally as the user is saying it, instead of trying to understand the utterance all at once.

Contrasted with the turn-based model, the incremental processing model is how humans process information (Aist et al. 2007). “Dialogue processing is, by its very nature, incremental. No dialogue agent (artificial or natural) processes whole dialogues, if only for the simple reason that dialogues are created incrementally, by participants taking turns at speaking” (Schlangen and Skantze 2011, p. 83). As a person recognizes words, they modify the conception of the current thought being expressed, and the thought can be understood even before its expression is finalized. Not only can incremental processing allow the IPA to respond quicker, people tend to prefer incremental systems over turn-based systems because they feel more natural (Kennington 2016, pp. 39-40). If a user gets stuck in the middle of a sentence, or if the system has acknowledged what the user intends to say to a high degree of certainty, the system can provide suggestions and feedback in a way that more closely resembles human dialogue. Some of these ideas are in use today, such as word suggestions when typing on a smartphone keyboard, and automatic speech recognition systems.

Incremental processing from the NLU lends the system to learn incrementally as well. Incomplete user utterances can be processed, preparing the memory model to retrieve bits of *high-level* data that can clue in on what the user is saying, and what they might say next. This

[2]Note that incremental processing is an entirely separate concept from the incremental memory requirement, but constructing the dynamic memory model with incremental processing will further enhance its incremental nature.

valuable information can be sent to other components of the dialogue system to enhance responsiveness and anticipate state changes.

Thirdly, flexibility is also desired in the model so that it can be adaptable. Like the data it manages, the model itself should be flexible to improvements so that it can meet the needs of its given domain. Since this model is not intended to be the solution to all of the problems discussed, it is sensible to use it as a starting point to gradually address them.

Lastly, data interpretability can help tremendously, as one will be able to observe, debug, and test more subtle aspects about how the data is organized. Interpretable data also ensures that the storage mechanism is not tightly coupled with the types of questions one may ask of it. This is opposed to opaque models in which stored data can only be interpreted at its output (e.g., neural networks). Instead, it should be feasible to construct multiple interfaces (that may use machine learning algorithms, for instance) to extract relevant features as necessary.

2.2 The Consequences of Dynamic IPA Memory

There are a some potential usage patterns that could be gained from enhancing the IPA with improved memory capabilities. In order of increasing complexity, these include (but are not limited to) (1) smarter databases, (2) improved context resolution, and (3) improved reasoning.

One immediate consequence of introducing dynamic memory is that the IPA is capable of functioning as a *natural language database*. Database systems are excellent for storing and retrieving structured information, but they require some technical knowledge of their query language. The dialogue nature of IPAs removes that requirement.[3] Users can thus request snippets of information that has been accumulating over previous dialogues with little technical knowledge. The model's dynamism would also handle new types of information and retrieve related data based on its related concepts.

Another effect is that users would need to repeat themselves less often. It is the common

[3]Natural language database implementations such as Quepy do exist (Andrawos et al. 2017), but are largely rule-based, which inhibits the system to learn new facts without developer intervention.

case that users of IPAs must repeat the steps of a dialogue to perform a task, even when there is a clear preference for some of the options selected. With improved memory, preferences could be learned over time to produce a shared context between user and system, enabling conversation to more naturally flow. This also leads to a deeper understanding of references. When a user refers to something, the system could use its internal context that it has been building throughout the conversation and resolve the reference. Related pieces of information could clue in on user intent when terms are ambiguous.[4]

A third outcome would allow for the system to reason about and organize related concepts. For example, suppose someone wishes to ask an IPA the question, “Who is my third youngest relative over the age of 30?”. Not only would the system need to resolve the context of terms like “my” and “relative”, it would need a conceptual model of a family tree, along with a reasoning engine that can deduce new relations from previously known facts. Context resolution in IPA systems has met moderate success, but beyond their limited history of facts, IPAs generally do not utilize high-level knowledge representations, let alone reason about them. Improved memory should come with the ability to represent information in a more complex network of related facts.

Memory is necessary for these consequences to occur, but not sufficient to guarantee that they will. Careful design decisions must be made, and the subsequent sections will build up a model that will achieve at least the first of these desired effects, the natural language database, with a direct path to expand into the others.

3 The Dynamic Memory IPA Model

A sufficient representation of the current knowledge state requires a robust memory interface that allows the IPA to not only store information about entities, but retrieve them just as easily. Although the DM does store and reason about the history of the dialogue, this kind of information is usually represented in a way that is inadequate for higher-level reasoning. This

[4]See Kennington et al. (2013) and Kennington (2016, pp. 123-159) for a model that explores this idea using an incremental, probabilistic model to resolve referenced visual objects.

history is typically meant to improve the capabilities of the dialogue system so that the DM can make more informed decisions, but is not normally able to be used for memory recall. To accommodate this, a new component in the DS is required, as shown in figure 2.

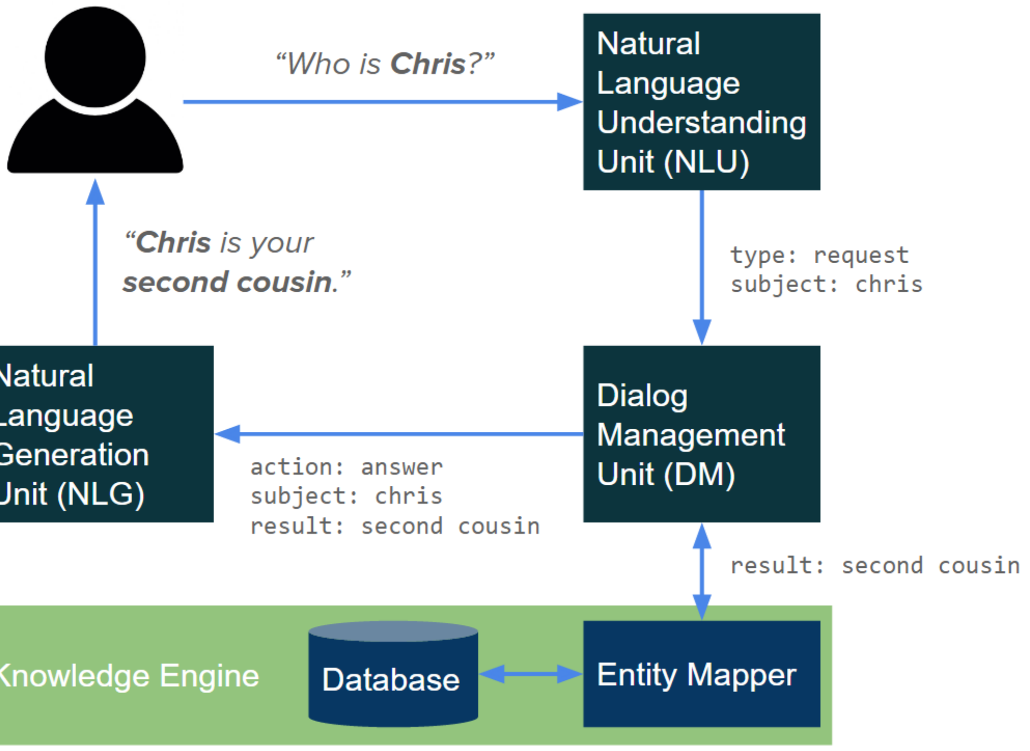


Figure 2: A Memory Enhanced Dialogue System

The knowledge engine is responsible for storing and retrieving information relevant to the DM's needs. This is accomplished with two subcomponents: a *database* for long-term storage of concepts related to entities, and an *entity mapper* that translates the needs of the DM to queries the database can understand. A connection is drawn to the knowledge engine via the DM so that it can retrieve personalized data for the user to answer related queries. Built this way, one immediate use-case as mentioned is to use this new component as a natural language interface to a database.

As shown in the diagram, the DM can capture uttered facts and then relay a subset of those facts back to the user. The utterance, "Who is Chris?" can be processed in the NLU to extract relevant information about the subject, in this case that "Chris" is the subject of the information request. Next, the DM issues a task for the knowledge engine to find pertinent information about the subject. The entity mapper takes care of issuing the corresponding query to the database (the architecture of which will be discussed in the next section), and returning the result back to the DM. The result is packaged up with the next DM task, in which the NLG outputs the answer, “Chris is your second cousin.”

The DM is not the only unit that could use the knowledge engine, however. The knowledge engine is intentionally kept separate from the rest of the dialogue system so that other services like the NLU and NLG could improve their understanding and generation based on the user’s structured content, in a way they previously could not. The goal of this model is to allow the dialogue system as a whole to gradually improve by benefiting from the easily accessible and rich data representation in the knowledge engine.

3.1 Database Technologies for Flexible Memory

To build a dialogue system that remembers facts about the user over the long term, one must be conscious of the substrate through which all those facts will be formulated. This means an appropriate storage mechanism which houses all the facts, i.e., database, must be chosen. The effectiveness of an IPA to recall information and reason about it relies heavily on how that information is represented.

> This dependence on representations is a general phenomenon that appears throughout computer science and even daily life. In computer science, operations such as searching a collection of data can proceed exponentially faster if the collection is structured and indexed intelligently. People can easily perform arithmetic on Arabic numerals but find arithmetic on Roman numerals much more time consuming. It is not surprising that the choice of representation has an enormous effect on the performance of machine learning algorithms (Goodfellow et al. 2016, p. 3).

Therefore, it is of paramount importance that the underlying choice of knowledge representation is flexible and robust enough to handle the complex behavior of natural language (or at least parts of it with ample room to expand).

Not only can performance be affected by data representation, but also the underlying thought process for how the entire dialogue system is built. If a database can store arbitrarily complex data, yet cannot retrieve any of it, it is useless. And if the database can only accept data types it has defined, it will be limited in capacity to learn about new data types. This does not bode well for a world in which information is constantly evolving.

One of the current ways IPAs can simulate "memory" is in their ability to perform web searches. This means submitting an API request to a search engine, such as Google Search, then responding back to the user with the top result. This gives the impression that the IPA contains its own vast repository of knowledge. So why doesn't the dialogue system store user utterances and build an index so the dialogue system can search it? In terms of information retrieval, search engines are too rudimentary. A search index is simply a giant collection of terms that point to the documents that contain them. This is useful for finding a larger body of resources from a smaller collection of search terms, but is not quite as useful for forming concepts about those terms. The system could point to where one could find information about 'banana peel', but could not incorporate it into its own dialogue dynamically. A search engine will retrieve an entire document, which is in most cases too large to be used for dialogue purposes. In effect, the IPA would only be useful in retrieving past dialogues as in, "On Monday, you said, 'My favorite color is orange.'" Thus, a more powerful storage mechanism is needed.

The maturest database architecture, the relational database, may be suitable for the underlying storage mechanism. The data is stored as a matrix; each row represents a concept or entity, while each column defines a property of an entity. Multiple tables (matrices) can be created to handle different types of concepts. Relationships between concepts can be searched by joining tables together on one or more conditions. This would allow the dialogue system to create any entities or relationships necessary. Unfortunately, while relational databases are fully capable of expressing arbitrary relationships, they suffer from two major deficiencies.

> For several decades, developers have tried to accommodate connected, semistructured datasets inside relational databases. But whereas relational databases were initially

> designed to codify paper forms and tabular structures——something they do exceedingly well——they struggle when attempting to model the ad hoc, exceptional relationships that crop up in the real world. Ironically, relational databases deal poorly with relationships (Robinson et al. 2015, p. 11).

The first problem with using a relational database for natural language processing is that a predefined schema must be provided in order to enforce that data is of a certain type. Incorporating any new types of data must involve changing the schema, an operation that will complicate interactions with the database tremendously. The second problem is that relationships between entities are never made explicit. One would have to recompute established relationships between tables every time a query is performed. Relational databases rely on `JOIN` expressions to find related information, which can consume resources inefficiently if used repeatedly. Relational databases are more suitable for *flat* data, where the number of possible relationships between entities are limited to a relatively constant size. This is fine for keeping track of user information like emails, but cumbersome for modeling complex language constructs like object descriptions.

Another option is to use a document-oriented database, in which data is structured as a tree. All components form a parent-child relationship, as seen in YAML, XML, and JSON serialization formats. The advantage over relational databases is that these structures can handle nested contexts, where nodes higher in the tree form more abstract concepts and relationships. In addition, a schema is not necessary, thus subtrees can be dynamically created and destroyed. However, these types of databases also have the difficulty of not being able to handle multiple relationships very well (Robinson et al. 2015, pp. 196-199). The constructed tree is effectively a single "view" into the database, which is a singular way of interpreting the data. There could be an equally valid, non-isomorphic tree that contains the relevant entities. As seen in relational databases, the same problem crops up, i.e., large numbers of relationships are costly to compute.

To handle the outlined data needs, a more free-flowing and flexible structure is required, where arbitrary relationships can be represented between any two entities. The archetypical data structure that models this type of behavior is known as a graph, and have been employed for use in graph databases. Graph databases offer a much more generalized way to represent data; its power comes from its ability to encode relationships between data points as edges in the graph.

Depending on their construction, graphs inside graph databases offer a much richer, multi-dimensional access point for data. The data structure is no longer constrained to a single pattern for accessing a group of entities. With graph databases, one can build up as many connections as possible, allowing an explosion of paths that one can find the data they want. Graph databases can be schemaless, allowing integration of evolving data types. New node and relationship types can be specified as they are encountered, forgoing the need to update a separate schema, as it updates in real time. In a relational database, this would be costly and require a lot of setup to accomplish. For this reason, graph databases are becoming popular in areas like social networks, where relationships between friends can be unbounded. Therefore, graphs seem like a reasonable abstraction for the type of data the dialogue system will need to store.

3.2 The Property Graph Model

There are several prominent methods to represent graph data, but the most popular graph databases use the **property graph model**, as seen in the Gremlin and Cypher query languages. According to Robinson et al. (2015, p. 206) a property graph has the following characteristics:

- It contains nodes and relationships.
- Nodes contain properties (key-value pairs).
- Nodes can be labeled with one or more labels.
- Relationships are named and directed, and always have a start and end node.
- Relationships can also contain properties.

Using a graph database, one can use it just the same as a relational database by treating each node as a row as in a relational database. The key-value properties are just like the accessible columns in a tuple. The real power lies within the relationships, which can link nodes (tuples) together with some defining characteristics. The following figure provides a common example of a typed graph. Each node represents an entity of type `Person`, with relationships being of type `KNOWS`. This graph models how each person it has defined knows each other person. Jim knows both Ian and Emil, but Ian only knows Emil at the moment. This graph can also be easily extended by with new relationships and entity types (e.g., Ian and Emil `WENT TO` an `Event`).

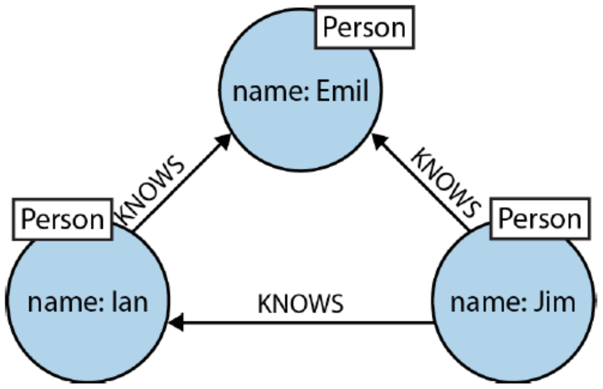


Figure 3: A Simple Graph Example, courtesy of Robinson et al. (2015, p. 28)

While graph databases improve flexibility and expressiveness, it is important to keep in mind that the flexible graph nature makes it easier for the data to become noisy or unwieldy. Incorrect abstractions could lead to bad data being returned, queries to perform poorly, or too many useless connections defined in the database. In that case, it would be beneficial to prune routes that create confusion/ambiguity, reducing the graph to a core set of features. The key is to enforce a way of structuring the graph so that it is as orderly as possible.

3.3 Ontology Engineering

To build a network of useful knowledge, a formal description of domain entities and relations between entities is necessary. This is referred to as an **ontology**, or in other words, “A controlled vocabulary for representing the types of entities in a given domain... [used in] the study of the basic categories of reality” (Arp et al. 2015, p. ix). In terms of a graph database, it is a specification of all the relationship and entity types in a graph, along with which relationships are allowed with which entities. Ontologies allow information to be reasoned about in a way that simplifies information overload by highlighting the essential relationships and entity types that build more complicated structures. A popular example of a lexical ontology is WordNet, which relates English words to their synonyms and definitions (Princeton University 2010).

The concept of building an ontology to allow information to be easily reasoned about is referred to as **ontology engineering** (Mizoguchi and Ikeda 1998). Improved ontology engineering would allow IPAs to store important facts within their knowledge representation and retrieve related information when necessary. As an example, if a user forgets the name of a person they interacted with previously, they may be able to describe this person to the system. If this person exists, the system can navigate their knowledge graph to pinpoint the answer. This is opposed to more traditional web search techniques, which are based on locating documents that match the user's query terms the most.

According to Russell et al. (2003), there has been limited success in producing a generalized ontology that would cover all possible types of knowledge. There have been several attempts to formalize knowledge into a logical model where the system should form new concepts on its own. However, this kind of a priori reasoning has been met with many issues.

> Several artificial intelligence projects have sought to hard-code knowledge about the world in formal languages. A computer can reason automatically about statements in these formal langugaes using logical inference rules. This is known as the knowledge base approach to artificial intelligence. None of these projects has led to major success... People struggle to devise formal rules with enough complexity to accurately describe the world... The difficulties faced by systems relying on hard-coded knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data (Goodfellow et al. 2016, p. 2).

One approach could have the dialogue system infer the relationships and entities using machine learning techniques; but this too poses a problem in which a copious (and sometimes infeasible) supply of training data is required. This is also problematic if the user introduces an entirely new domain that the training data did not address. These issues are possible to overcome, but requires a steep difficulty curve. It is easier to forgo machine learning techniques on understanding the knowledge in favor of a simpler stepping stone which can be used to progress into more involved models.[5]

Instead of dealing with the challenge of deciphering the meaning of all entities and relationships, a simpler approach would have the knowledge engine not needing to know the purpose of the data types at all. The memory could be user-driven, where instead of providing an interface where the system gets to decide what types of relationships are important, a user-driven approach would have the user specify the important relationships themselves. The user is not restricted to a subset of items, but rather can conform their information to whatever domain they please. This lifts the burden off the developer needing to explicitly define what each entity and relationship is supposed to represent.

The user will be able to specify and highlight the important entities and relationships, and how they should interact together. This is not unlike a user organizing their own filesystem, placing files into their own respective folders, and arranging them to best suit their use-case. But instead of using a GUI or a command-line, the dialogue system should be able to translate the user's high level descriptions of the structure into what these descriptions might represent. Thus, the ontology should be used to encourage the user to provide guided descriptions of the data they are either entering or looking for.

This is an ambitious task, yet it is not insurmountable, as graph databases offer a rich set of pattern matching tools that can simplify the necessary logic. Similar to how a search engine matches *words* in documents, graph databases match graph *relationships*. In both approaches, there is no need to understand the contents of the entire resource to make it useful. Rather, it is the structural patterns that can be exploited.

Once the ontology is defined, it is the job of the entity mapper to enforce it. This can be as simple as defining an API that shuffles data back and forth without much processing. This could be improved in subsequent iterations, where graph traversals could extract a richer set of data an

[5]This model is left open to use machine learning techniques to extend it, but will not be core to its function for the purposes of this paper.

create new relationships based on old ones, much like a reasoning engine. But what determines what should be defined in the graph, and what should be left up to the reasoner to figure out on its own? “A tension that exists in graph modeling is what to make explicit (structure) and what to infer through traversal (process). The trade-off is between, like much of computing, space and time” (Rodriguez 2011). A careful balance between graph structure and graph processing should be optimal. For the purposes of this paper, the pattern matching capabilities of the graph database will more than suffice.

3.4 A Simple Graph Ontology

Having the necessary background in understanding the dynamic memory model’s overall function, this section will provide a first iteration on this model which promotes the construction of a dialogue system that can represent simple concepts in a ready-to-use memory representation. But before it is given, some assumptions must be made about the dialogue domain.

1. Information is *entity-centric*, i.e., there exist entities and each of them have one or more properties that annotate the entities’ current state.
2. Entities can form simple direct relationships with other entities.
3. Duplicate properties are not allowed, but instead are shared among entities that have them.

Item 1 restricts the representable dialogue concepts to discrete entities with their own properties. This is partly due to humans commonly reasoning and referring to objects in this way, and also partly because the property graph model (as mentioned previously) provides all the necessary constructs to represent this information without difficulty. Item 2 allows for entities to be defined in terms of other entities (e.g., the attic is *above* the kitchen). This can be useful for forming spatial, temporal, or abstract relations. Lastly, item 3 is where this ontology can become quite useful. If a property like *red* is uniquely identified in the graph, all objects which have that property will be pointing to the same instance in memory. This makes it easy to search for entities that have certain properties, as it is as simple as finding the set intersection of all entities that have direct links to those properties. Not only is this computationally efficient, but it also clearly identifies ambiguities. If multiple entities are matched in an expression that assumes just one (e.g., "Find me a video titled Macaroni" returns two videos), then the graph will be able to identify the ambiguity and notify the appropriate modules. These ideas will be explored in a more concrete example later on.

Progressing to the formal property graph definition, there will be two concepts (node types) defined in the graph: (1) an `entity` (e.g., a person, event, physical object), and (2) one or more `properties` or components of that entity (e.g., a name, address, time). These concepts are related together with two relationship types: (1) the `entity relation`, which relates two entities together (e.g., John knows Emily), and (2) the `property relation`, which relates an entity to a property (e.g., the banana is yellow). The `entity` node contains a single text field which indicates its type so that different types of entities can be differentiated easily. Likewise, the `property` nodes and `entity relation` contain a single text field to indicate the property type and relation type respectively.

These node and relationship type definitions form the basis for the meta-ontology in which the user can adapt their own domain specific ontology within. Figure 4 shows an abstract example of this ontology. Entities in green form relations (in blue) to other entities. Each entity can be associated with zero or more properties, which may be shared.

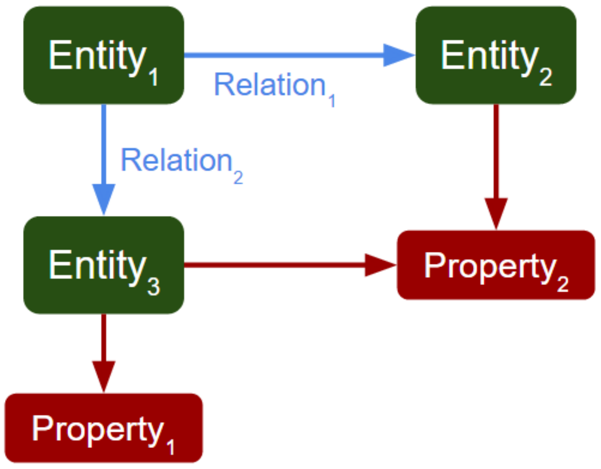


Figure 4: An Abstract Ontology Example

Note that this ontology definition deliberately removes explicit hierarchical relationship between entity types. An entity may have several types of relationships with other entities, but whether a type forms a subtype of another type is always implicit within the data contained within a node or relation. The statement, “all animals are organisms,” cannot be expressed as part of the top level ontology of the graph. Firstly, this is to ensure that the data does not restrict itself from committing exceptional cases or types that have changed their meaning over time. Secondly, this is to encourage the logic in the entity mapper to be as simple as possible to interface between the database and the rest of the dialogue system.

3.5 A Simple Graph Ontology Example

To see the practicality of such an ontology, suppose the knowledge engine has been fitted to the domain of keeping track of relationships between people in a social network. People are represented as entities, who can have `met` relationships with other people. Suppose each person has three properties: `first name`, `last name`, and `email`. A user can update this information by notifying any information updates to the application’s IPA (dialogue system). As the dialogue progresses, the graph may form new `met` relationships, or delete old ones. New entities of type `person` can be created with their associated name properties. Having the necessary information in place, a user can then request information about other people by referring to them by their first or last name. Many questions can be translated directly to a graph traversal, such as “Give me everyone who knows Sam,” or “What’s Skinner’s first name?”

What would happen if a user requests the email of a person with the first name of “Chelsie,” when there are two people with that name? Figure 5 shows how this may be structured in the graph database.

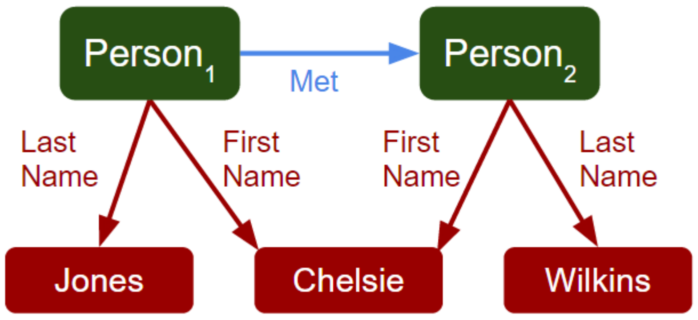


Figure 5: Ontology Example 1

There exist two people in the graph, one named “Chelsie Jones”, and the other named “Chelsie Wilkins”. Notice that the `Chelsie` property node is merged into one. As mentioned earlier, not only does this prevent duplication and to save space, the graph structure allows one to easily find out all the entities that share a first name `Chelsie`. In this graph, there are just two, but in a scenario where there could be hundreds to millions of people, the graph could have many people that share certain first names, last names, or other properties.

An utterance passes to dialogue system, through the NLU, and `Chelsie` is recognized as a named entity. The DM receives this input, and goes on to try to resolve this named entity by querying the knowledge engine. The knowledge engine’s entity mapper submits a query to the database to find any properties that have `Chelsie` as its content. The node is found, and the entities associated with that node is returned (`Person 1` and `Person 2`).

But there is a problem: who is the user talking about? `Chelsie` is ambiguous. Thus, the system must perform disambiguation, which can involve either (1) the NLU/DM keeping track of extra contextual clues to make one choice most probable, or (2) the DM asking the user to resolve the ambiguity before proceeding. Option 2 is the easiest to implement, but if done correctly option 1 can make the dialogue interactions feel more natural.

Assume that the dialogue system cannot resolve this particular ambiguity on its own. The DM/NLG asks, “Are you referring to Chelsie Jones, or Chelsie Wilkins?” Depending on the user’s answer, the DM will resolve the reference and retrieve the appropriate email.

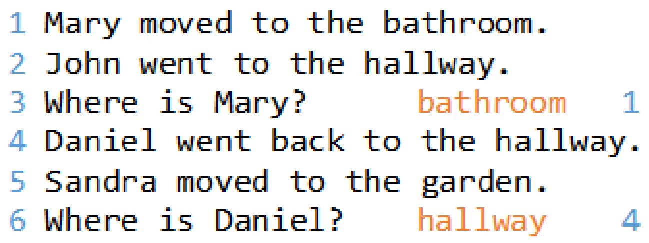
4 Software Implementation

To demonstrate the abilities of the dynamic memory model in a concrete scenario, a software prototype has been implemented in Python.[6] The corpus to evaluate the model's performance is supplied by the Facebook bAbI Tasks Data v1.2 (*The bAbI project* 2015).

As seen in Figure 6 the data is laid out with one sentence per line. Each sentence can either be a statement or a question:

- **Statement** – Provides facts about the situation.
- **Question** – Requests an answer pertaining to the preceding statements. The correct answer is provided alongside the question.

This project uses Task 1 Single Supporting Fact, in which each statement describes four characters moving between six different rooms. The questions periodically ask the room in which a person is currently in, and the objective is to answer them all correctly. Due to the nature of the model, training will not be necessary to answer each question. Therefore, the entire document will be used for test evaluation.



```
1 Mary moved to the bathroom.
2 John went to the hallway.
3 Where is Mary?       bathroom   1
4 Daniel went back to the hallway.
5 Sandra moved to the garden.
6 Where is Daniel?     hallway    4
```

Figure 6: bAbI Task 1, Single Supporting Fact

Note that this dataset provides a small complication to the above mentioned graph ontology; there must be some facility in place to keep track of episodic relationships between individuals and rooms (as opposed to assuming all information is temporally independent). This issue will be discussed when constructing the graph in the methods section below.

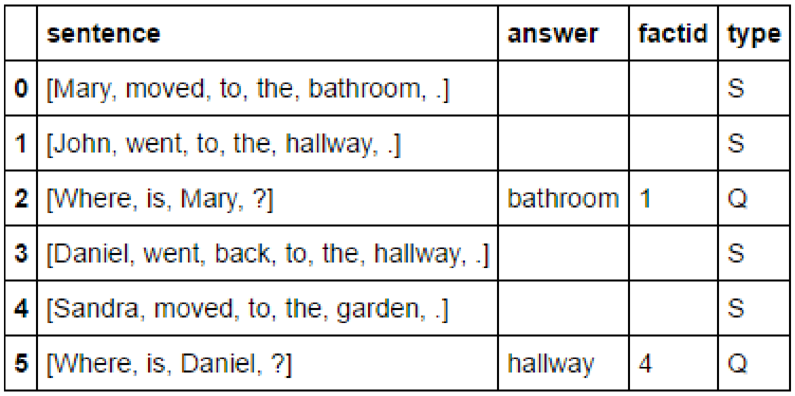
[6]See https://github.com/Hyperparticle/graph-nlu

4.1 Methods

All code is written in Python using a Jupyter notebook to display all intermediate results. Pandas DataFrames provide simple visualization and data processing, NLTK's tokenizer and part-of-speech tagger extract entities by tokenizing sentences (i.e., splitting them into individual words) and tagging each word with its part of speech (e.g., noun, verb, adjective etc.), a Neo4j database (and driver) supplies graph storage and efficient querying capabilities, and the Scikit Learn library computes an accuracy score.

4.1.1 Text Processing

In the first step, Pandas to imported `qa1_single-supporting-fact_train.txt` from the corpus into a DataFrame. Since the system will require no training, this 3,000 line document will be used as the test set. Once imported, each sentence is tokenized using NLTK's word tokenizer to produce an array of words in the `sentence` column, answers supplied for questions are provided in the `answer` column (it is blank if the sentence is a statement), and the sentence type (S or Q) column indicates whether the sentence is a statement or question.[7] This frame is shown in figure 7.



	sentence	answer	factid	type
0	[Mary, moved, to, the, bathroom, .]			S
1	[John, went, to, the, hallway, .]			S
2	[Where, is, Mary, ?]	bathroom	1	Q
3	[Daniel, went, back, to, the, hallway, .]			S
4	[Sandra, moved, to, the, garden, .]			S
5	[Where, is, Daniel, ?]	hallway	4	Q

Figure 7: The processed DataFrame

To make it easier to reason with these sentences, the first step is to extract the relevant

[7]The `factid` column can be safely ignored.

entities from each statement and question. Due to the simplicity of the data, each statement can be thought of as a (`subject`, `relation`, `object`) triple, which lends itself quite nicely to build relational information in a graph. To do this, define a function S, that when given a sequence of statement tokens, produces this triple. For instance,

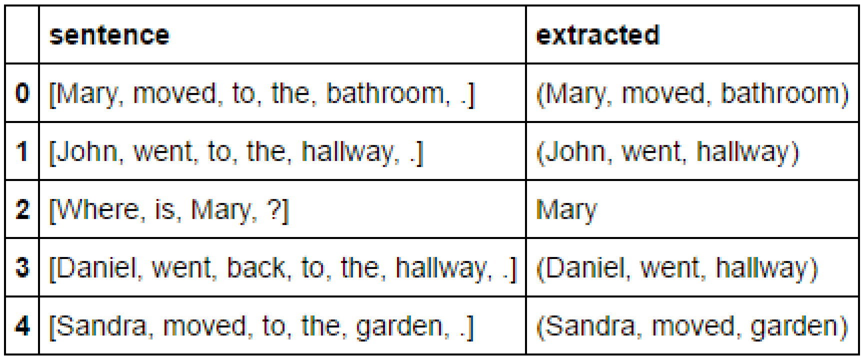
$$S([Mary, moved, to, the, bathroom, .]) = (Mary, moved, bathroom)$$

To find the appropriate words that correspond to the subject, relation, and object in this dataset, one can use their parts-of-speech. NLTK's part-of-speech tagger tagged each token in each sentence and stored it in a column called `tag`. If there is a word tagged as a singular proper noun (NNP), it is the subject, if there's a verb (VBD), it is the relation, and if there is a singular noun (NN), it is the object. This triple can then be merged with other triples to form a graph of relationships between individuals and rooms (as will be seen below).

Likewise, to test the graph define another function Q, that when given a sequence of question tokens returns the entity that the question is asking for.

$$Q([Where, is, Mary, ?]) = Mary$$

The output of this function can be used to construct a query on the graph to find the room that answers the question. Figure 8 shows the final output of the S and Q functions.



	sentence	extracted
0	[Mary, moved, to, the, bathroom, .]	(Mary, moved, bathroom)
1	[John, went, to, the, hallway, .]	(John, went, hallway)
2	[Where, is, Mary, ?]	Mary
3	[Daniel, went, back, to, the, hallway, .]	(Daniel, went, hallway)
4	[Sandra, moved, to, the, garden, .]	(Sandra, moved, garden)

Figure 8: Extracted Data

4.1.2 Graph Construction

The second step was to construct queries to merge the statement triples in the graph database, and request information from the database given a question object. Having started the Neo4j graph database, a connection can be established with the Neo4j Python driver, which allows for Cypher (language) queries to be executed in the database.

One of the first ideas that one may come across when building the graph is to represent the subject and object as nodes, and the relations as edges between them. In effect, individuals' names and the rooms that they were in are all the entity types in this domain, with no properties required. In the Cypher query language, this is represented as

$$(subject) - [relation] \rightarrow (object)$$

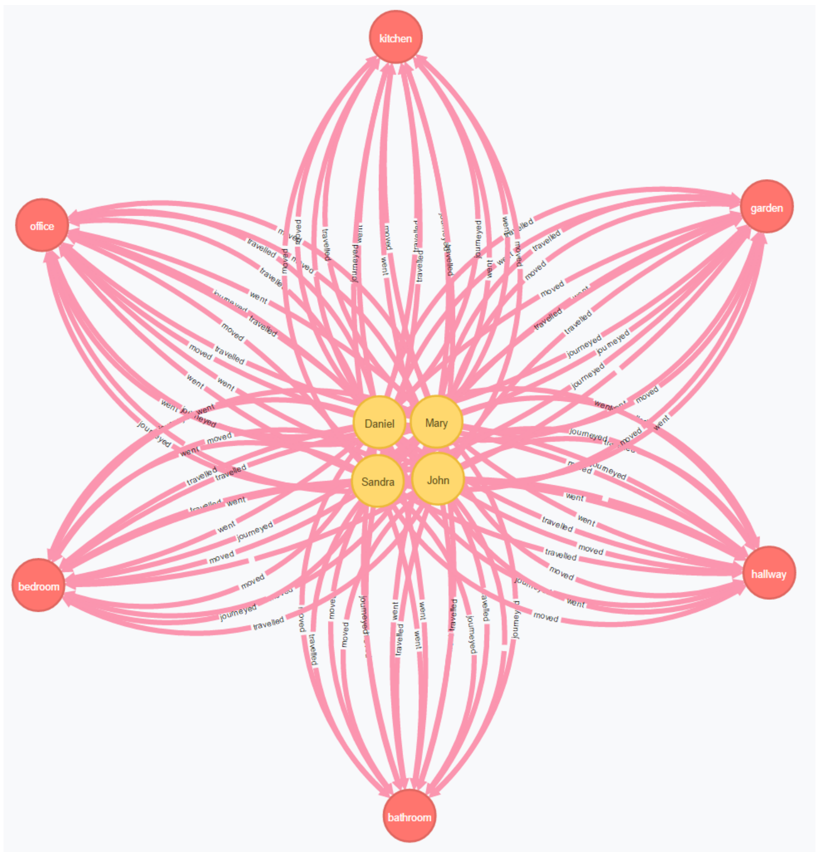


Figure 9: Direct Relationship Graph

Figure 9 illustrates the results of merging the subjects and objects together. The rooms are represented as red nodes, names of people are given as yellow nodes, and verb relations are shown as the arrows in magenta.

The graph is a reasonable first start, as the relations point each person to where they have been. But this poses a potential problem: how could the system know where each person is right now, or where they have been previously? All that can be known from the graph is which rooms a person has been in, because they may have visited them all multiple times. From the figure, every person has been to every room. Because the information in the corpus is episodic, the graph ontology will need to be modified slightly to accommodate temporal relationships between states.

A second approach is to ensure that the relations can be marked in the graph. Since edges cannot be drawn between other edges, there is no easy way of representing state transitions between relations in this knowledge representation. Instead, by placing the content of a relation as another node type in the graph, edges can be used to highlight relationships between verb relations. This is done by forming a linked list of events, where each event corresponds to a person updating the room that they are in. For each newly formed relation node, an edge is drawn from the head of the event list to this node. This means that the head of this list will always be the most recent relation created in the graph, and hopping backwards in this list will show the order in which rooms were visited (for one person).

This can be expressed as

$$(subject_{n-1}) \to (relation_{n-1}) \to (object_{n-1})$$

$$(subject_n) \to (relation_n) \to (object_n)$$

$$(relation_{n-1}) \to (relation_n)$$

where subjects and objects can be shared, but relations are always unique.

Figure 10 displays an updated version of the previous graph. Blue nodes are individuals' names, red nodes are names of rooms, and green nodes are relations (which used to be edges in the previous graph). The yellow edges between green nodes always point to the next action that a person took, and the gray edges link each relation with their subject and object.

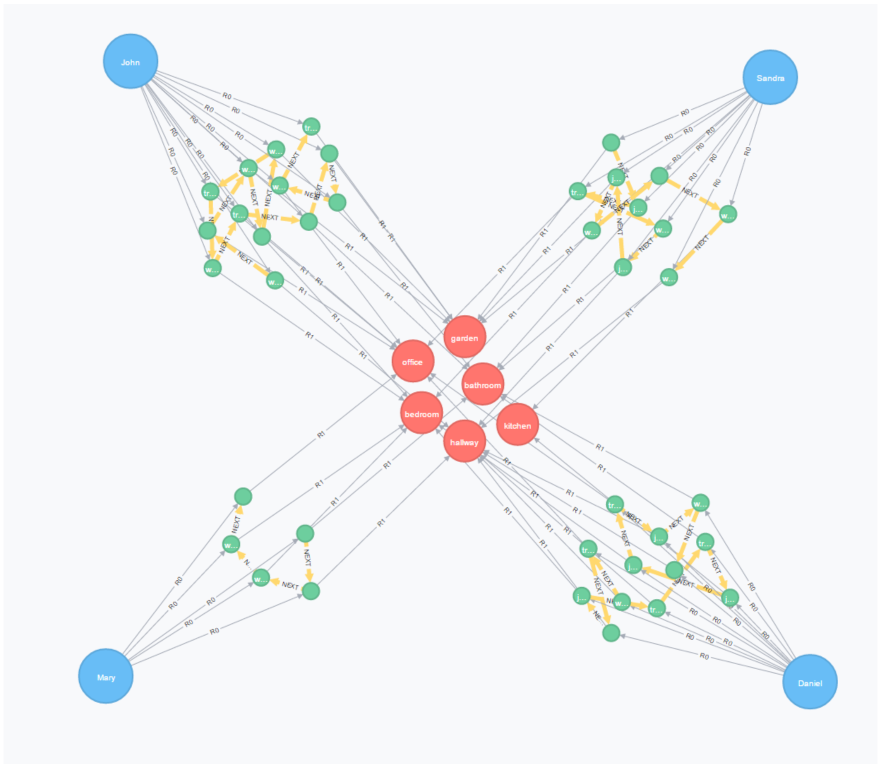


Figure 10: Linked List of Relations Graph

Now the system can query the graph for useful information, e.g., to answer the question, “Where is Mary?” As before, NLTK tokenized and tagged to produce the entity under discussion of “Mary,” then Pandas inserted the entity in a Cypher query to match against the room node such that there is a relation connecting that room with the “Mary” node, and that this relation is the head node (has no next element) in the event list.

Thus, a single query has been constructed such that it can find the answer to every question in the test set. But the system can do better: since the graph has been constructed with a history of states, not only can this system find the room that a person is currently in, the system can also find the entire history of rooms that a person has been in, listed in reverse chronological order. A more advanced question is, “Where has John been recently?” This can be done in just one graph query that traverses the linked list, collecting rooms/relations as it goes along to previous states.

4.2 Evaluation

Having the queries to update and retrieve information finalized, the final step is to evaluate the accuracy of the system's text processing and querying. This involves iterating over all the statements and questions, updating the graph with the extracted (`subject`, `relation`, `object`) triples, and querying the graph based on the entity in each question to produce an array of predicted answers to all questions. Scikit Learn compared against the actual answers to all questions, resulting in an accuracy of 100%.

Because this dataset is so simple, the accuracy should be expected to be around 100%, so this result should be of no surprise. But that is not the only aspect of this system that should be considered. The data in the constructed graph is much richer than just the information for which room a person is in. It also provides the name of each person, the name of each room, and a complete history of past dialogue states in relation to names and rooms. But since the graph does not explicitly know about rooms or people, it could work with any sentences of the form (`subject`, `relation`, `object`).

The most important result of this software implementation is that this system (1) does not require any training data to operate effectively, (2) updates its internal knowledge incrementally, and (3) provides very interpretable, structured high-level concepts that can be used by any service that needs them.

5 Conclusion

In summary, this paper has shown the following:

1. The importance of improving IPA memory.

2. Proposed the dynamic memory model that serves as an abstract framework for approaching the memory problem.

3. Explained the technical design considerations for such a model, such as introducing the

knowledge engine consisting of an entity mapper and graph database, and a simple graph ontology for representing user utterances.

4. Demonstrated a proof-of-concept software implementation by evaluating a simple domain.

IPAs are useful software agents, as natural language is both expressive and concise, making dialogue suitable for executing tasks pertaining to some user need. However, their understanding capacity is stunted in large part by their inability to retain vital concepts in memory as the dialogue progresses. The dynamic memory IPA model has been provided as an approach to this problem with the goal of being dynamic, incremental, flexible, and interpretable. By incorporating a knowledge engine consisting of a graph database to a traditional dialogue system, this model is able to structure and store high-level concepts efficiently and provide an access point for other modules in the system to leverage (and update) this information. Furthermore, this model takes advantage of the property graph model in graph databases, defining a simple entity-centric ontology which shows how user-driven semantic information can be constructed both to find entities based on the properties they may have and to identify ambiguities. Finally, a Python implementation of this approach has been evaluated with the Facebook bAbI tasks dataset showing that the dynamic memory model can be a viable approach to accomplishing its goals due to the flexibility that graph databases provide.

This work has been a very brief introduction to understanding how interaction could be stored in memory to be leveraged for future use. More work is necessary to ensure that this model will be suited for more complicated domains, such as providing the reasoning required to complete bAbI tasks 2-10 (*The bAbI project* 2015). In addition, there has yet to be a comprehensive answer to how the knowledge engine can be utilized to its full potential for modules such as the NLU, DM, and NLG so that the IPA's dialogue system can perform optimally. Online evaluations should be conducted to further identify the aspects of the the model that are a benefit or hindrance to user interactivity.

References

Aist, G., Allen, J., Campana, E., & Gallo, C. G. (2007). Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods. *Decalog 2007*, 149.

Amazon. (2016). The Alexa Prize. Retrieved from https://developer.amazon.com/alexaprize

Andrawos, E., Berrotarán, G. G., Carrascosa, R., Alemany, L. A. i., & Durán, H. (2017). Quepy. Retrieved from https://github.com/machinalis/quepy

Arp, R., Smith, B., & Spear, A. D. (2015). *Building ontologies with basic formal ontology*. Mit Press.

Clark, H. H. (1996). *Using Language* (B. G. Blount, Ed.). Cambridge University Press. doi:10.2277/0521561582

Clark, J. (2016). Electricity Bill With DeepMind-Powered AI. Retrieved from https://www.bloomberg.com/news/articles/2016-07-19/google-cuts-its-giant-electricity-bill-with-deepmind-powered-ai

Fillmore, C. J. (1981). Pragmatics and the description of discourse. *Radical pragmatics*, *3*, 143–166.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

IBM. (2016). AI Xprize. Retrieved from http://ai.xprize.org/

Kennington, C. (2016). *Incrementally resolving references in order to identify visually present objects in a situated dialogue setting* (Doctoral dissertation, Bielefeld, Universit{ä}t Bielefeld, Diss., 2016).

Kennington, C., Kousidis, S., & Schlangen, D. (2013). Interpreting Situated Dialogue Utterances: an Update Model that Uses Speech, Gaze, and Gesture Information. In *Sigdial 2013* (August, pp. 173–182).

Linn, A. (2016). Historic Achievement: Microsoft researchers reach human parity in conversational speech recognition. Retrieved from http://blogs.microsoft.com/next/2016/10/18/historic-achievement-microsoft-researchers-reach-human-parity-conversational-speech-recognition/%7B%5C#%7Dsm.0000yiw0tgaondlbpwx2n98ro0h1j

Lison, P. (2014). *Structured Probabilistic Modelling for Dialogue Management* (Doctoral dissertation, University of Oslo). Retrieved from http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf

Martin, J. H. & Jurafsky, D. (2000). Speech and language processing. *International Edition*, *710*.

Mizoguchi, R. & Ikeda, M. (1998). Towards ontology engineering. *Journal-Japanese Society for Artificial Intelligence*, *13*, 9–10.

The bAbI project. (2015). Facebook. Retrieved from https://research.fb.com/downloads/babi/

Actions on Google. (2017). Retrieved from https://developers.google.com/actions/

Amazon Developer. (2017). Retrieved from https://developer.amazon.com/alexa

Pichai, S. (2016). I/O: Building the next evolution of Google. Retrieved from https://googleblog.blogspot.com/2016/05/io-building-next-evolution-of-google.html

Princeton University. (2010). About WordNet. Princeton University. Retrieved from http://wordnet.princeton.edu

Reddit. (2016). Remind me this weekend. Retrieved from https://www.reddit.com/r/SiriFail/comments/57w4yv/remind%7B%5C_%7Dme%7B%5C_%7Dthis%7B%5C_%7Dweekend/

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: new opportunities for connected data*. " O'Reilly Media, Inc."

Rodriguez, M. (2011). Knowledge Representation and Reasoning with Graph Databases. Retrieved from https://markorodriguez.com/2011/02/23/knowledge-representation-and-reasoning-with-graph-databases/

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (2003). *Artificial intelligence: a modern approach* (3rd ed.). Prentice hall Upper Saddle River.

Schlangen, D. & Skantze, G. (2011). A General, Abstract Model of Incremental Dialogue Processing. In *Dialogue & discourse* (Vol. 2, *1*, pp. 83–111). Retrieved from http://www.elanguage.net/journals/dad/article/view/361/1452