Proceedings of the Twenty-Third Conference of the Association for the Advancement of Artificial Intelligence (AAAI-08), Chicago, IL, July 13-17, 2008.

Automating To-Do Lists for Users: Interpretation of To-Dos for Selecting and Tasking Agents

Yolanda Gil and Varun Ratnakar

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
gil@isi.edu, varunr@isi.edu

Abstract

To-do lists have been found to be the most popular personal information management tools, yet there is no automated system to interpret and act upon them when appropriate on behalf of the user. Automating to-do lists is challenging, not only because they are specified as free text but also because most items contain abbreviated tasks, many do not specify an action to be performed, and often refer to unrelated (personal) items. This paper presents our approach and an implemented system to process to-do list entries and map them to tasks that can be automated for the user by a set of agents. Since the format of to-do entries is not very amenable to natural language processing tools that can parse and create a structured interpretation, our approach is to exploit paraphrases of the target tasks that the agents can perform and that specify how the free-text maps to the task arguments. As users manually assign to-do to agents for automation, our system improves its performance by learning new paraphrases. We show an evaluation of our approach in a corpus of 2100 to-do entries collected from users of an office assistant multi-agent system.

Introduction

To-do lists are the most widely used of all personal information management tools, used more than calendars and contact lists (Jones and Thomas 1997). Many to-do list managers are widely available on the web (e.g., todolistsoft.com, tadalist.com, todoist.com, voo2do.com among many others). Although there have been user studies of to-do lists that have uncovered use patterns and desiderata (e.g., (Bellotti et al 2004; Hayes et al 2003)), no system has been developed to date to provide intelligent assistance to users. As (Norman 1991), we see an immense and largely unexplored opportunity through automatically interpreting, managing, executing, and in general assisting users with to-do lists. To-do lists include items that can be acted upon by intelligent agents that have the capabilities to accomplish some of those items. This kind of investigation becomes possible as agent-based systems are developed to assist users with routine tasks, such as office assistants (e.g., Electric Elves (Tambe et al 2002) and CALO (Myers et al 2007; Berry et al 2006)) and other assistants that exploit agents and services on the web (e.g., (Berners-Lee et al 2001)).

Our initial focus is on mapping to-do list entries into the capabilities of agents that can automate those entries for the user. To-do entries are jotted by the user as free text, while agent capabilities are formal descriptions that typically include a task to be accomplished and its arguments. A study of to-do lists created by several users of the CALO system over time showed that to-do lists have idiosyncratic content that make their interpretation a challenge (Gil and Ratnakar 2008). Less than one in seven entries could be automated by some agent, since many entries contain tasks that only the user can do. Of those, most were incomplete in that they did not specify necessary arguments. Two thirds of the entries in that study did not begin with a verb. In summary, to-do list entries are not well-formed or complete sentences typically found in text, and lack the dialogue context that is present in conversational interfaces (e.g., (Allen et al 2001)).

An important goal of our research is to accommodate new tasks as new agents become available or existing agents extend their capabilities. This poses an important requirement that the system can learn to map to-do entries into new agent capabilities. Learning would have the added advantage that the system could adapt its mappings to suit the idiosyncrasies of particular users (such as personal abbreviations). Another advantage of learning is that the system would be able to take on any domain, rather than being designed and optimized for specific domains or purposes.

This paper describes an approach and implemented system that uses paraphrases of agent capabilities to map the free text of a to-do entry into a capability and its arguments. The system learns new paraphrases as it sees examples of correct mappings, which can be collected from users as they task the agents by hand, or acquired from volunteer contributors (Chklovski 2005; Chklovski and Gil 2005).

The paper begins with a general description of the different aspects of to-do list interpretation and mapping. After describing our approach and implementation, we present an evaluation of the system using a corpus of to-do list entries collected from users over several months.

Assisting Users with To-Do Lists

We consider six phases in interpreting and automating to-do list entries, shown in Figure 1. The left side shows the kind of result expected from each step, and the right side gives an example. As we will see, the automation of each of these steps can be turned into an interface extension that can be useful to the user in its own right.

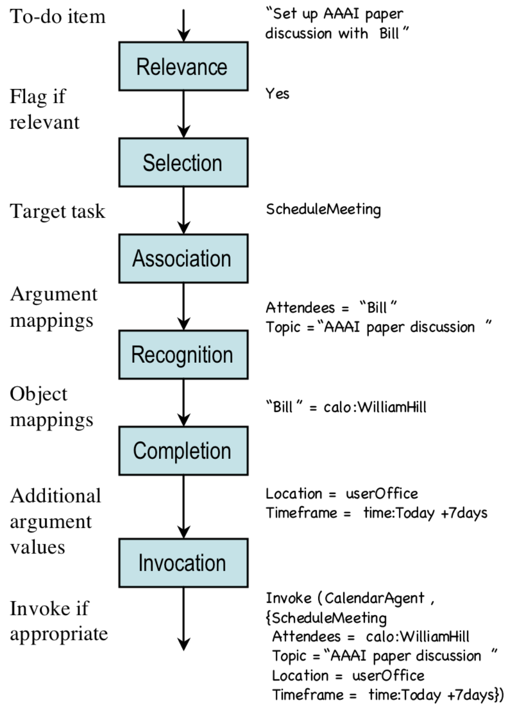
Relevance involves determining whether a to-do list entry is relevant to any of the agent capabilities. That is, whether the to-do entry is within the scope of the tasks that the system can automate for the user. If the system has good performance in this step, it can help the user by highlighting those entries in the to-do list so that the user is aware that there are agents that could automate them.

Selection involves finding the agent capability (or target task) that is appropriate for automating a given to-do entry. Note that the system may generate several candidates. If the system has good performance in this step, the top ranked candidate can be shown to the user as a possible way to automate the entry. Other candidates, and there should not be too many, can be shown as alternatives to the user as a pull-down menu.

Association will determine which chunks of the to-do entry text correspond to which arguments in the target task. Good performance in this step would allow the system to have the menu to task the agent pre-filled by those mapped arguments, requiring little or no editing from the user to task the agent.

The last three phases are shown here for completion but have not been the focus of our work to date. **Recognition** finds an object in the system that corresponds to the text chunk of an argument. For example, the to-do entry may contain the first name of a person and the recognition phase would find that person's formal representation, which will allow the agent internally to access email addresses or other information useful for the task. Notice that not all arguments can be mapped to formal objects. An example is the topic of a meeting, which is only useful (at least in our work) to the other human attendees. **Completion** involves completing the other argument values not included in the to-do list entry, either by using default values, typical values learned for each user, or values inferred from the user's context (e.g., other meetings, emails, etc.) **Invocation** involves deciding whether the agent should be invoked, which can be done with the user's explicit consent or using some measure of adjustable autonomy for the task.

These last three stages could be done by the agent, where the agent could have mechanisms to set default values and to make decisions about when to act. The



To-do item
"Set up AAAI paper discussion with Bill"
Relevance
Flag if relevant
Yes
Selection
Target task
ScheduleMeeting
Association
Argument mappings
Attendees = "Bill"
Topic = "AAAI paper discussion"
Recognition
Object mappings
"Bill" = calo:WilliamHill
Completion
Additional argument values
Location = userOffice
Timeframe = time:Today +7days
Invocation
Invoke if appropriate
Invoke (CalendarAgent , {ScheduleMeeting
Attendees = calo:WilliamHill
Topic = "AAAI paper discussion"
Location = userOffice
Timeframe = time:Today +7days})

Figure 1: Phases involved in interpreting and automating to-do list entries. This paper focuses on the first three phases.

first three stages are properly the responsibility of a to-do list manager and are the focus of this work.

Using Paraphrases to Interpret To-Do List Entries

Our approach to interpreting to-do lists is to use paraphrases to re-state the to-do list entry until it can be mapped to a target task or the system runs out of paraphrases to try. We pursued this direction for two main reasons. Because the to-do entries are incomplete and not well-formed sentences, they do not lend themselves to natural language techniques such as chunking and parsing. The second reason is that we want to improve the system's performance by learning as users use it.

In our approach, the system maintains a list of paraphrase patterns for each target task. A paraphrase essentially reflects how the user might invoke certain tasks. Paraphrase patterns contain special keywords combined with task arguments. Some example paraphrase patterns for a task are:

Task: SendEmail +recipient-s +topic
Paraphrases:
  ask +recipient-s to +topic
  message to +recipient-s
  send +topic to +recipient-s
  email +recipient-s about +topic

In this example, *"SendEmail"* is the target task, which has the two arguments *"recipients"* and *"topic"*. Task arguments are marked with a "+" in the paraphrase pattern. Keywords in the patterns include "ask", "to", and "about." The keywords are used for matching paraphrases with the entry, while the arguments are used to map portions of the to-do entry to those arguments. In order to have a successful match, all the keywords in the pattern should match the to-do entry in the order that they are written in the paraphrase. Any portion of text in between matched keywords and marked in the pattern as an argument gets bound to that argument. Each paraphrase pattern has a direct correspondence between its arguments and the arguments in the target task. Lists are marked in the pattern with the suffix "-s" at the end of the argument, which is matched with special keywords such as "and", "or", "/", "," in the to-do entry. Using the paraphrase patterns listed above, these are some example to-do items and their matches:

Ask Bill to give feedback on paper
-> ask +recipient-s="Bill" to +topic="give feedback on paper"
Ask Bill for feedback on paper
-> --No match--
Message to Bill about project report
-> message to +recipient-s="Bill" +topic="project report"
Send message to Bill
-> message to +recipient-s="Bill"

Note that the to-do entry *"Send Message to Bill"* matches the paraphrase pattern *"message to +recipient-s"* as the whole pattern is contained within the entry. In other words, the text matches a paraphrase pattern if all keywords in the pattern match the text. The reverse is not the case, for example the to-do entry *"Email to Bill"* would not be matched with the paraphrase pattern *"send email to +person"* since not all the pattern keywords are present in the to-do entry.

In selecting a target task, there may be several candidates and we produce a ranking based on the match. We define the specificity of the paraphrase as the number of keywords in the paraphrase. In case of a match, the ranking of the match is the specificity of the paraphrase pattern that it matched against.

Paraphrase patterns can be provided by hand, which may be useful to seed the system and bootstrap its initial performance. Our system can learn paraphrase patterns as the user invokes an agent by hand by selecting an agent from a menu and tasking it through a form-filling interface (this is how users task agents with to-do items in the current system). An alternative approach is to learn paraphrases from volunteer contributors where the system can acquire the paraphrases off-line from volunteers by prompting them for variants of already known paraphrase patterns (Chklovski 2005).

To learn paraphrases, we create a variabilized pattern of each to-do entry that is mapped to a target task, where the variables correspond to the arguments in the target task. Our training examples come from the user as they invoke agents by hand. Other researchers have addressed paraphrase learning from text corpora (e.g., (Barzilay and Lee 2003)), but make use of vast amounts of data (in the order of megabytes) that we do not have available. Learning is currently done by looking at the set of "answers" that are collected from the user over time. These answer keys contain the to-do entry and the matching task and arguments as filled by the user. The system scans through the to-do entry and replaces the text mapped to the argument with the task argument variable to create the paraphrase pattern. An example answer key is:

to-do-entry: email John about demo deadlines
task-type: SendEmail +recipients +topic
arguments: recipients="John" topic="demo deadlines"

which would result in the following learned pattern:

email +recipients about +topic

However, not all paraphrases that are created are eligible for use. We restrict paraphrase patterns as follows. There needs to be at least one keyword, so for example "*+topic*" is not allowed. This restriction is to avoid matching anything and everything. The more specific the paraphrase, the better chance there is of getting a correct match. Another restriction is that arguments must be separated by one or more keywords. For example, "*ask +recipients +topic*" is not allowed. This second restriction is needed because we do not have a recognition phase. That is, the system does not have a mechanism to correctly identify where the first argument stops and the second argument starts. In our current system, that paraphrase pattern would not be allowed (or learned). We discuss this restriction further below.

Metrics for Evaluation

Since our system is the first of its kind ever developed (at least to our knowledge), we needed to define metrics to measure its performance. Because of the novelty of this research area, we have focused on evaluating the performance of the system's functionality in terms of its internal algorithms rather than on evaluating usability issues. Therefore, we evaluated the system off-line, with a corpus of to-do entries collected from CALO users over several months.

The evaluation tests performance on identification, selection, and association. As we discussed earlier, good performance in each of the tasks can have direct utility to the user.

Metrics for Task Relevance Identification

For task relevance identification, we measure performance with accuracy, precision, and recall metrics. The accuracy metric is defined as the percentage of to-do items that were correctly flagged or not flagged. Precision is defined as the fraction of to-do entries that were flagged as relevant that are actually relevant:

Precision = {to-dos relevant to agent capabilities AND flagged as relevant}

/ {to-dos flagged as relevant}

Recall is defined as the fraction of to-do entries actually relevant that are actually flagged by the system as relevant:

Recall = {to-dos relevant to agent capabilities
AND flagged as relevant}
/ {to-dos relevant to agent capabilities}

Precision and recall can be combined as a joint F-measure metric that takes their harmonic mean, F-metric = 2/(1/Precision + 1/Recall).

Metrics for Task Class Selection

To evaluate performance with respect to task class selection, we want to use metrics that reflect the system's value to the user. Precision and recall would not be appropriate. For example, if an option is generated, but is ranked 17th, the system's precision and recall would be high because the correct selection would have been generated. But since the user may never see it, the performance metrics that we use should reflect that it was not found useful to the user. Therefore, improving on the precision and recall metrics should not be our ultimate goal. Ideally, the system should always aim to choose the right task as its top-ranked suggestion. To measure performance in this respect, we use the following metric:

t-metric = % entries where the correct answer is the top option proposed

To track the system's behavior, we use the following metrics as well:

g-metric = % entries where the correct task mapping is proposed as a candidate

c-metric = average number of candidates generated that are selected to appear in the top k to show user

Note that performance in the g-metric and the c-metric directly affects the t-metric, but our goal is to improve the t-metric performance.

Metrics for Argument Association

For association, we measure performance in terms of how many corrections would the user have to do to the initial mapping and bindings proposed by the system in order to get the desired argument mappings. That is, a given suggestion will be used to fill the initial task description that the user will correct if needed. Our metric is the *edit distance*, namely how many changes would the user need to make to the system's mapping and bindings if the user had to correct the top suggestion.

Evaluation Methodology

For our evaluation, we obtained to-do entries and processed them off-line. These to-do entries were obtained as users were jotting them as part of their normal use of the system. The system was completely passive and provided no assistance with the to-do lists. Our goal at this stage of the project is to assess the quality of the system before it is deployed for use with true on-line learning and assistance.

Evaluation Corpus. We collected a corpus of 2400 to-do list entries from a dozen users of CALO during two subsequent annual evaluation periods that lasted several months each. A subset of 300 entries were extracted as a reference corpus and used for development and analysis purposes. The remainder 2100 entries were set aside for evaluation. When providing examples throughout this paper, we created fictional to-do entries to protect the privacy of our users.

Target Tasks. The CALO Task Ontology where agents can register their capabilities is under development, and has not been populated. However, it contains a set of 87 task categories that were used to evaluate the CALO system. These task categories only have the class name, and no arguments are specified. Most of them are not relevant to the to-do list corpus collected. Example task categories are: Decide, Explain, Move, Calculate, Execute, Commit, Learn, Affect, Wait, Input, Borrow, Claim, AnswerWithChoice, AnswerWithData, Announce, and Invoke. We selected a subset of these tasks that we could map reasonably well to capabilities of agents available in CALO. We represented their arguments based on the kind of information that automating these tasks would require for these agents. This resulted in a set of 18 candidate target tasks that we used for evaluation:

PlanTrip +location
PlanTrip +event +date +location
ScheduleVisit +person +timeframe
ScheduleInterview +visitor +timeframe
ScheduleSeminar +person +timeframe
ScheduleMeeting +attendees +topic +timeframe
RescheduleMeeting +attendees +timeframe +topic
ScheduleGroupVisit +group +topic +timeframe
Buy +equipment +person
Buy +office-supplies +person
Buy +flight-reservation +person
Lookup +document
Download +url
Print +document
Edit +document
Approve +purchase-request
Approve +travel-request
SendEmail +recipients +topic

Answer Set. We asked an annotator to label the test set with the correct answers for each task. In some sense only the user that created the to-do item would know what the correct answers were based on their true intention and context, so the annotator took a best guess at what the tasks were intended to accomplish. For example:

To-Do-Entry: Setup discussion on IUI paper
Correct-Selection: ScheduleMeeting +person +timeframe +topic
Correct-Association: topic = "IUI paper"

The to-do entries that did not correspond to any task were given NULL selection and association labels.

There were a total of 382 entries that were marked with a target task, the rest did not correspond to any task. For the entries that were mapped to tasks, 130 were mapped to *SendEmail*, 90 to *Lookup*, 58 to *Edit*, 51 to *ScheduleMeeting*, 25 to the *PlanTrip* tasks, 10 to the *Buy* tasks, 9 to *ScheduleInterview*, 3 to *ScheduleVisit*, and 2 each to *Download*, *Print* and *ScheduleSeminar*. There were four tasks that did not appear at all in the annotated reference corpus and were the *RescheduleMeeting*, *ScheduleGroupVisit*, *Buy*, and *Approve* tasks.

Notice that the tasks included in each of the sets are not completely unrelated to one another in that they have many terms and argument types in common. Therefore, the system must discriminate among them correctly.

Results

We compared the performance of our system with respect to a baseline system that made random decisions, as well as with several configurations of bag of words matching. Our system is shown as **ordered matching**, to distinguish it from bag of words matching of paraphrase patterns. We show learning curves for each metric at 8%, 15%, 30%, 60% and 100% of the training data using 4-fold cross validation.

The baseline system performs no NL processing. For the relevance phase, this baseline system flags an entry as relevant 14% of the times, since that is the proportion of relevant entries estimated by prior corpus analysis (Gil and Ratnakar 2008). For the selection phase, it chooses from the set of target tasks one or two tasks at random, and randomly ranks them. The baseline system does not generate any associations.

We also compared the performance of our system with simpler systems with minimal natural language processing capabilities. These systems use simple bag of words matching between the to-do entry and the task paraphrase patterns. We built two variants. One variant normalizes words from their morphological variations, using the well-known Porter stemmer (artarus.org/~martin/PorterStemmer). The second variant also uses synonyms from WordNet (synsets), in addition to using the stemmer.

Results for Task Relevance Identification

The accuracy metric is shown in Figure 2. Our system has very good performance, over and above the performance of the other methods, with 87.6% accuracy after learning. After learning, out of 2100 entries our system classified correctly 1841 entries (flagged 160 as relevant that were actually relevant and did not flag 1681 that were actually not relevant) and misclassified the rest (flagged 37 that were actually not relevant and did not flag 222 that were actually relevant). The accuracy of a simple baseline system that always guessed relevance would be 18% for this dataset (a bit higher than the 14% relevance found for the corpus in (Gil and Ratnakar 2008)). A simple baseline system that always guessed

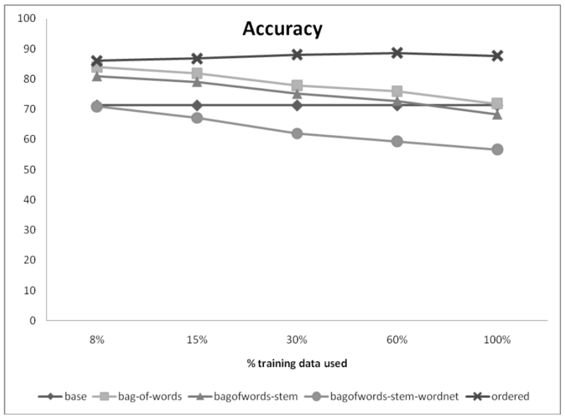


Figure 2: Task Relevance Identification: Accuracy

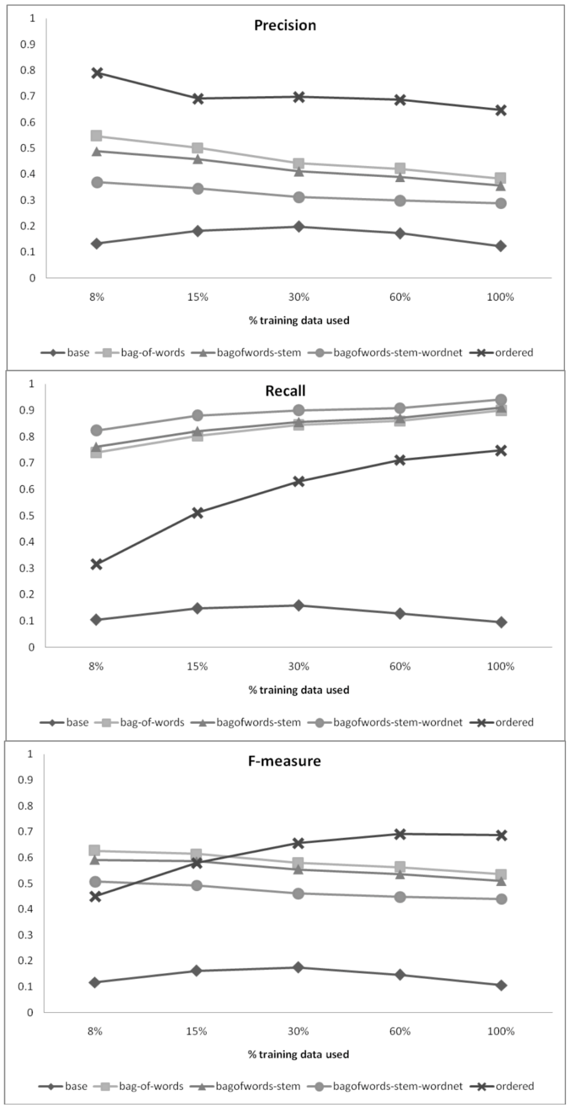


Figure 3: Task Relevance: Precision, Recall, F-Measure

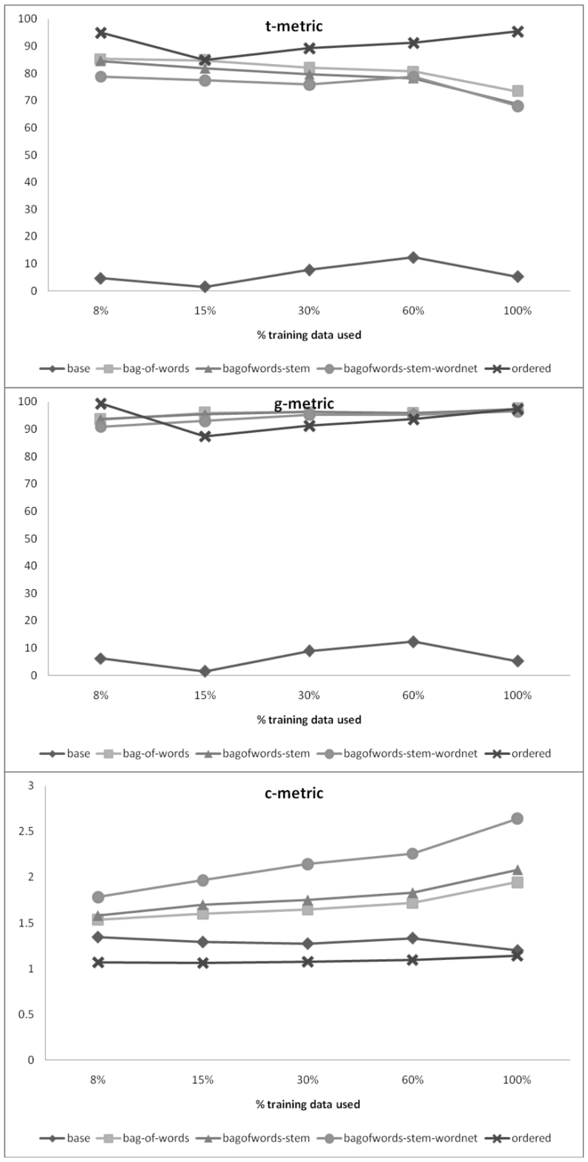


Figure 4: Task Selection metrics: t-metric, g-metric, c-metric

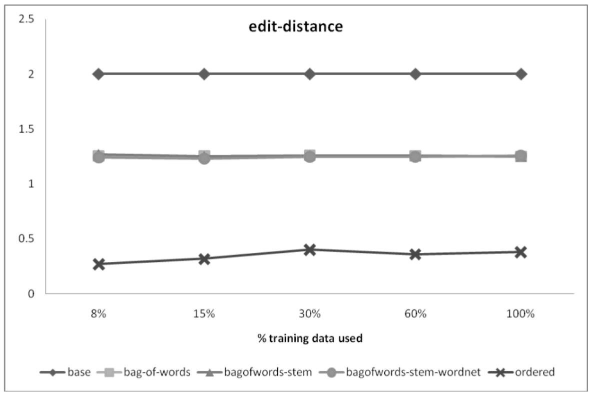


Figure 5: Argument Association: edit distance

irrelevance would be 82% accurate but to no good use as it would never do anything to assist the user.

Figure 3 shows the precision, recall, and F-measure metrics. Our system shows very high precision, significantly higher than the other methods from very early on the training. This means that when our system intervenes to make suggestions about a to-do entry, it is right to do so with very high probability. Recall is low for our system, as can be expected since its matches are much more restrictive than the other methods. We discuss how to address our overly constrained matching in the discussion section below.

Results for Task Class Selection

Figure 4 shows the data for task class selection. Our system performs best at the task, which is reflected in the t-metric results, and reaches over 90% correctness early during training. Although the g-metric shows that all methods are able to generate the right answer as part of their candidate set, the c-metric shows that our system generates a single candidate very early on during training. The user would be shown the correct target task with very high probability.

Results for Argument Association

Figure 5 shows the results for argument association. Notice that the baseline and bag of words methods do not have the ability to make such associations, so for those methods we show the amount of parameters that the correct answer would contain. This corresponds to how many edits the user would have to make. For example, *"Email Bill"* would correspond to *sendEmail recipient="Bill"*, and since the to-do does not contain a topic then the user would only have to fill one argument of the target task by hand. In addition, we only reflect the edit distance of to-do entries where the selection phase for the method ranked the correct answer at the top. Our system not only does this automatically but performs extremely well, requiring between 0.2 and 0.4 user edits on average.

Discussion

We noted that the performance of our system suffered from a limitation of our paraphrase pattern language, namely that all arguments in a paraphrase must be separated by keywords. An example is:

Todo Entry: Send Bill final paper

Desired pattern: Send +recipient-s +topic

Ideally, such paraphrase pattern could be handled if the system were able to detect chunks in the to-do entry and map them to each of the arguments. In this case, the system could identify *"Bill"* and *"final paper"* as chunks and map each to the respective argument. This would also help distinguish between the two *Buy* tasks (one for equipment and another for office supplies). Off-the-shelf chunking and parsing tools are difficult to use in to-do entries because to-do entries are not complete or well-formed sentences. We will investigate the use of adaptive parsers in future work.

An alternative that we are planning to explore is to expand our paraphrase approach to the recognition phase. That is, to collect paraphrases for ontological objects that can give us type information. For example, we would collect paraphrases such as *"Bill"* and *"Bill H."* that refer to the ontological object calo:WilliamHill which is of type calo:Person. By combining recognition and matching, the system would be able to use paraphrase patterns without separating keywords.

Conclusions

This paper introduced a novel use for AI techniques in assisting users with to-do lists, which are used for personal information management more than calendars and other tools. By interpreting to-do entries and mapping them to the capabilities of agents, we can assist users by automating and managing their to-do lists. To-do lists are challenging to interpret as they are often incomplete and abbreviated. Our approach is to exploit paraphrases of agent capabilities to interpret the entry and identify the task arguments. We discussed the implementation of this approach in the context of the CALO office assistant architecture. Our system improves its performance by learning new paraphrases from users.

One important empirical question is whether users would adapt the way they jot to-do entries to facilitate the system's interpretation and quality of assistance. To investigate this, we are planning to test our system in an interactive mode.

Acknowledgements

We gratefully acknowledge funding for this work by DARPA under contract no. NBCHD030010. We thank Tim Chklovski for many useful discussions in the early stages of this work. We also thank Karen Myers and Ken Conley for useful feedback on this work and its integration with the CALO architecture.

References

Allen, J.; Byron, D. K.; Dzikovska, M.; Ferguson, G.; Galescu, L.; and Stent, A. 2001. "Towards Conversational Human-Computer Interaction." *AI Magazine*, 22(4):27-37.

Barzilay, R. and Lee, L. 2003. "Learning to Paraphrase: An Unsupervised Approach Using Multiple-Sequence Alignment." *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL/HLT)*.

Bellotti, V., B. Dalal, N. Good, P. Flynn, D. Bobrow, N. Ducheneaut. 2004. What a To-do: Studies of Task Management Towards the Design of a Personal Task List Manager. *ACM Conference on Human Factors in Computing Systems (CHI)*.

Berners-Lee, T., Hendler, J., and Lassila, O. 2001. "The Semantic Web." *Scientific American*, May 2001.

Berry, P.; Conley, K.; Gervasio, M.; Peintner, B.; Uribe, T.; and Yorke-Smith, N. 2006. "Deploying a Personalized Time Management Agent." *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Chalupsky, H., Gil, Y., Knoblock, C. A., Lerman, K., Oh, J., Pynadath, D. V., Russ, T. A. and M. Tambe. 2002. "Agent Technology to Support Human Organizations." *AI Magazine*, Vol. 23, No 2.

Chklovski, T. 2005. "Collecting Paraphrase Corpora from Volunteer Contributors". *Proceedings of the Third International Conference on Knowledge Capture (K-CAP)*.

Chklovski, T. and Gil, Y. 2005. An Analysis of Knowledge Collected from Volunteer Contributors. *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*.

Gil, Y., and Ratnakar, V. 2008. "Towards Intelligent Assistance for To-Do Lists." *Proceedings of the 2008 International Conference on Intelligent User Interfaces (IUI)*.

Hayes, G., Pierce, J. S., and Abowd, G. D. 2003. "Practices for Capturing Short Important Thoughts." *ACM Conference on Human Factors in Computing Systems (CHI)*.

Jones, S. R. and Thomas, P. J. 1997. "Empirical assessment of individuals' 'personal information management systems'." *Behaviour and Information Technology* 16(3): 158-160.

Myers., K., P. Berry, J. Blythe, K. Conley, M. Gervasio, D. McGuinness, D. Morley, A. Pfeffer, M. Pollack, M. Tambe. 2007. "An Intelligent Personal Assistant for Task and Time Management." *AI Magazine*.

Norman, D. A. 1991. Cognitive artifacts. In J. Carroll, editor, Designing Interaction: Psychology at the Human-Computer Interface. Cambridge University Press.