Computer-generated Dialog as a Resource-bounded Activity

Toby Donaldson and Robin Cohen
{tjdonald,rcohen}@uwaterloo.ca
Department of Computer Science
University of Waterloo

Introduction

Resource bounds abound in conversation. Conversants must work under time pressure, they have working memory limitations, and they often have incomplete or uncertain information about the state of the conversation. Fortunately, conversation can tolerate errors and misconceptions: if you do not understand something the speaker says, then you can initiate a clarification dialog, or, if you recognize an error on the speaker's part, you can interrupt them to propose a repair.

In this abstract, we discuss some past work on resource-bounds in conversation and describe our own work on modelling conversation as a dynamic constraint satisfaction problem.

Resource Bounds in Conversation

The most immediate and common resource bound in conversation is time. In real-time human-human conversation, there tends to be very few pauses, and people must often speak before they have settled upon "just the right thing" to say. Dialog is thus a time-pressured activity. Carletta (1992) proposes that time-pressure can be combatted by taking conversational risks. An interesting example of such "risk" is evidenced by *self-repairs*, where a speaker catches himself in mid-utterance, e.g. *Go to the left — uh — the right of the swamp.*. This research has motivated the development of a systemic natural language generation system that is capable of anytime language processing (Carletta *et al.* 1993). The system is seeded with default values for all possible output, and so if the system is interrupted prematurely, it can always say something by using its defaults, or by purposely inserting a *hesitation* if a certain level of solution quality has not been achieved. Fillers can easily be taken from a list of stock phrases and hesitations are usually preferable to causing a long pause, or giving a default answer that would have serious consequences if incorrect.

Time is not the only resource bound in conversation. Humans have working memory limitations, and (Walker 1994) shows that limited working memory affects a discourse strategy's utility. In particular, her computer experiments show that informationally redundant utterances help agents handle their memory limitations. Importantly, Walker demonstrates that the efficiency, or quality, of a dialog depends on the complexity of the task. Minimizing the number of utterances is important, but it is not the only concern in dialog; better dialogs can sometimes result by slightly increasing the length of the dialog through the careful insertion of redundant utterances.

Carletta and Walker's work both show that discourse behaviour changes significantly when resource bounds are taken into account. Their work contrasts with much computational discourse research, which has usually not taken resource bounds into consideration, but instead has assumed that agents will have all the time and memory they need to decide what to say. However, with an increasing focus on real-time dialog systems using voice (e.g. Mostow *et al.* 1994), GUIs (e.g. Rich & Sidner 1996), and virtual reality (e.g. Rickel & Johnson 1997), resource bounds must be dealt with. While Walker and Carletta *et al.* both focus on modelling resource bounds in *humans*, they are interested in developing practical systems that can interact with humans in a natural way. Such work is a good first step towards practical systems, but ultimately we want to develop discourse systems that are both sensitive to human resource limitations, and take full advantage of the computer's superior speed and more accurate memory.

Constraint-based Discourse Agents

Our research focuses on developing a model for turn-taking in discourse which can be applied to the design of intelligent agents that interact with users (or possibly other agents). The model is designed to specify how and when to take a turn in a discourse regardless of whether the form of communication is natural language or something more visually oriented (e.g., GUIs). Previous research in discourse processing has examined how to analyze utterances produced by other conversational participants and how to generate appropriate cooperative responses, e.g. (Chu-Carroll & Carberry 1994; van Beek, Cohen, & Schmidt 1993).

Conversation is ultimately a real-time process, so our

model explicitly takes time constraints into account. We treat a conversational agent as consisting of three separate and concurrent processes:

- A *pre-processing module* that takes raw speech (or key clicks, or mouse movements, etc.) as input and converts that into a form the agent can reason with;
- A *thinking module* that solves a *turn-taking problem*; the solution to a turn-taking problem is effectively the system's next utterance;
- A *listening module* that looks for the appropriate turn-ending and turn-yielding signals from the current speaker. An agent needs to know if its current solution is of high enough quality to be used, but it needs to know if the current speaker is willing to give up the floor.

This model does not require any one particular kind of implementation strategy, although we are interested in modelling complex interaction as a kind of dynamic constraint satisfaction problem (DCSP) that can be solved by local search methods. At any time in a conversation, an agent will have some number of goals to consider, and one (or more) must be chosen to be achieved. More concretely, at any one time, an agent is considering which goal to act upon from a list of goals $G_1, \ldots, G_k$. The problem is for the agent to find the n (usually $< k$) goals that can be placed in the order that results in the fewest penalty points. A fixed number of slots, labelled $V_1, \ldots, V_n$ are used, and it is known that when the agent is required to act, it will first try to achieve the goal in slot V_1, and then V_2, and so on. Ordering constraints can be put on pairs of goals, such that if two constrained goals are out of order, a penalty is incurred. A perfect ordering would have 0 penalty points. We model this problem as a DCSP, which is a natural and flexible framework for representing such an ordering problem. It is dynamic since, new goals/constraints may be added/dropped, based on what the other conversant says and how the conversation progresses.

It is important to note that we are assuming a *local search* method will be used to solve the DCSP. Classical backtracking methods could be used (Tsang 1993), but local search is preferable for at least three reasons: local search algorithms are usually anytime algorithms; local search methods work on over-constrained CSPs without modification; and, when the structure of the DCSP changes, i.e. a constraint or goal is changed, then the best solution found for the previous CSP can be used to seed the current CSP.

As mentioned, the constraints are between pairs of goals, and they generally refer to the types of goals. For example, *repair* type goals should usually be satisfied before *information-seeking* goals, because the repair could affect the correctness of the information being sought. Other constraints based on time, causality, user preferences, the domain, etc. can also be taken into consideration. More details on the particular DCSP we are solving and the relevant constraints can be found in (Donaldson & Cohen 1997a; 1997b).

Example Dialog

What follows is a simplified example of how the DCSP framework works. Suppose a student goes to a course advisor for help choosing a schedule of courses. The student begins by asking

STUDENT: *I'm a part-time engineering major. Can I take CS100 for credit?*

As the student is speaking, the advisor is listening and we assume that *question* type goals are triggered in the advisor. Question-type goals are less specific than the goal-types given in (Donaldson & Cohen 1997b), but they are adequate to handle the current example. We suppose that 4 yes/no question goals are triggered roughly in the order given:

- G_1: "Are you in electrical engineering?" (electrical engineering students cannot take CS100)
- G_2: "Are you minoring in mathematics?" (math minor students cannot take CS100)
- G_3: "Do you mean CS100 on-campus?" (CS100 is an ambiguous name)
- G_4: "Have you taken a CS course for credit?" (you cannot take CS100 if you've taken any other CS course)

The problem is to put $G_1, \ldots, G_4$ in the order that minimizes the length of the dialog. For this example, the only concern is to answer the question posed by the user; moreover, the answer to G_3 does not alter the ultimate advice to the student. Goals G_1 and G_2 are triggered because the system believes by default that an engineering or math student must have already taken a CS course. Question G_4 "covers" G_1 and G_2, and we can see that it is the best question to ask first. The idea of minimizing the number of questions asked was first used by van Beek *et al.* (1993) to direct the design of clarification subdialogs when inferring a user's plan. The DCSP formalism we represent here is more general, since it deals with arbitrary constraints between general turn-taking goals, and allows for anytime processing. Without being able to fully reason about the set of possible goals, the system could, for instance, produce the dialog below (based on the goal ordering G_1, G_2, G_3, G_4):

ADVISOR: *Are you in electrical engineering?*
STUDENT: *No.*
ADVISOR: *Are you minoring in mathematics?*
STUDENT: *No.*
ADVISOR: *Do you mean CS100 on-campus?*
STUDENT: *Yes.*
ADVISOR: *Have you taken a CS course for credit?*
STUDENT: *Yes.*
ADVISOR: *Since you've already taken a CS course, you cannot take CS100 for credit.*

For example, the student might speak so quickly that the advisor has no chance to perform any goal-ordering while the student is speaking. Advising sessions are not typically highly time-pressured situations, but examples do arise where a student will deliver a lot of information quite quickly; in this situation, the advisor might hesitate in order to buy some more thinking time. We could follow the suggestion of Carletta et al. and decide when to hesitate instead of trying to achieve the first goal, by checking if the goal-ordering is above a domain-dependant quality threshold θ_{hesitate}.[1] Then, if the system is forced to act[2] and the solution quality is below θ_{hesitate}, it can hesitate by saying, for instance, *Let me think about that...*, in the hope that in the time it takes to say this an acceptable goal ordering will be found. If the system has *not* found a better goal ordering, then it can hesitate again, although eventually the system should give up and say something like *I'm sorry, could you re-phrase your question?*. Now suppose that in our example the quality of the ordering G_1, G_2, G_3, G_4 is above θ_{hesitate}. In this case, question G_1 would be asked, and the advisor would keep ordering the goals while the exchange continues, so in that time it could come to achieve the better order of G_4, G_2, G_3. It would then ask G_4 and G_2, so in retrospect the system acted as if it had found the suboptimal, but adequate, ordering G_1, G_4, G_2, G_3 at the beginning.

Using DCSPs also allows us to handle the case where the student realizes what the advisor is getting at and so cuts a subdialog short by directly providing the sought-after information. In our sample dialog, if after the systems asks G_3 the student happens to answer his own original question and responds *Oh, I see, I can't take CS100 because I've already taken CS102*, then the advisor can drop all of its goals with respect to answering the student's question, ending the dialog.

Discussion

Carletta *et al.* (1993) suggest another possibility of for taking advantage of a performance profile: along with a lower quality bound of θ_{hesitate}, an upper quality bound of θ_{speak} can be used to let the system decide when to speak in cases where it can speak, but is not absolutely required to do so. Such cases arise when a speaker issues a *turn-yielding* signal, that is, an indication that they might be willing to give up the floor. Because conversation is a dynamic activity where new information is coming in all the time, we do not expect that using θ_{speak} will prove useful. In general, it appears to be better to wait until the circumstances of the conversation require you to speak, in order to gather all the information you can for deciding what to say. Also, interrupting a speaker is a delicate issue, since too many interruptions from a computer system can irritate a person to the point where they want to turn the system off.

We are also interested in the idea of using a "working memory" of goals to help make the system act efficiently. At any one time, the system has n goals in memory, but only k ($< n$) goals are put in working memory. Local search is applied to working memory, and a separate and concurrent process is used to decide which goals in "long term" memory should be promoted to working memory. For large numbers of goals, this scheme could prove more efficient than performing local search on all goals.

[1] An ordering with fewer penalty points is said to be of higher quality.

[2] For instance, if the current pause length is unacceptable.

References

Carletta, J.; Caley, R.; and Isard, S. 1993. A system architecture for simulating time-constrained language production. Technical Report rp-43, Human Computer Research Centre, University of Edinburgh.

Carletta, J. 1992. *Risk-taking and recovery in task-oriented dialogue.* Ph.D. Dissertation, University of Edinburgh Dept. of Artificial Intelligence.

Chu-Carroll, J., and Carberry, S. 1994. A plan-based model for response generation in collaborative task-oriented dialogues. In *Proceedings of AAAI-94*, 799–805.

Donaldson, T., and Cohen, R. 1997a. Constraint-based discourse agents. In *AAAI-97 Workshop on Constraints and Agents.*

Donaldson, T., and Cohen, R. 1997b. A constraint satisfaction framework for managing mixed-initiative discourse. Presented at AAAI-97 Symposium on Mixed-Initiative Interaction.

Mostow, J.; Roth, S.; Hauptmann, A.; and Kane, M. 1994. A prototype reading coach that listens. In *Proceedings of AAAI-94*, 785–792.

Rich, C., and Sidner, C. 1996. Adding a collaborative agent to graphical user interfaces. In *UIST'96.*

Rickel, J., and Johnson, L. 1997. Integrating pedagogical capabilities in a virtual environment agent,. In *Proceedings of the First International Conference on Autonomous Agents.*

Tsang, E. 1993. *Foundations of Constraint Satisfaction.* Academic Press.

van Beek, P.; Cohen, R.; and Schmidt, K. 1993. From plan critiquing to clarification dialogue for cooperative response generation. *Computational Intelligence* 9(2):132–154.

Walker, M. 1994. Experimentally evaluating communicative strategies: The effect of the task. In *Proceedings of AAAI-94*, 86–93.