A Hierarchical POMDP Approach to Dialogue Management

Presented to the Robot Learning Lab

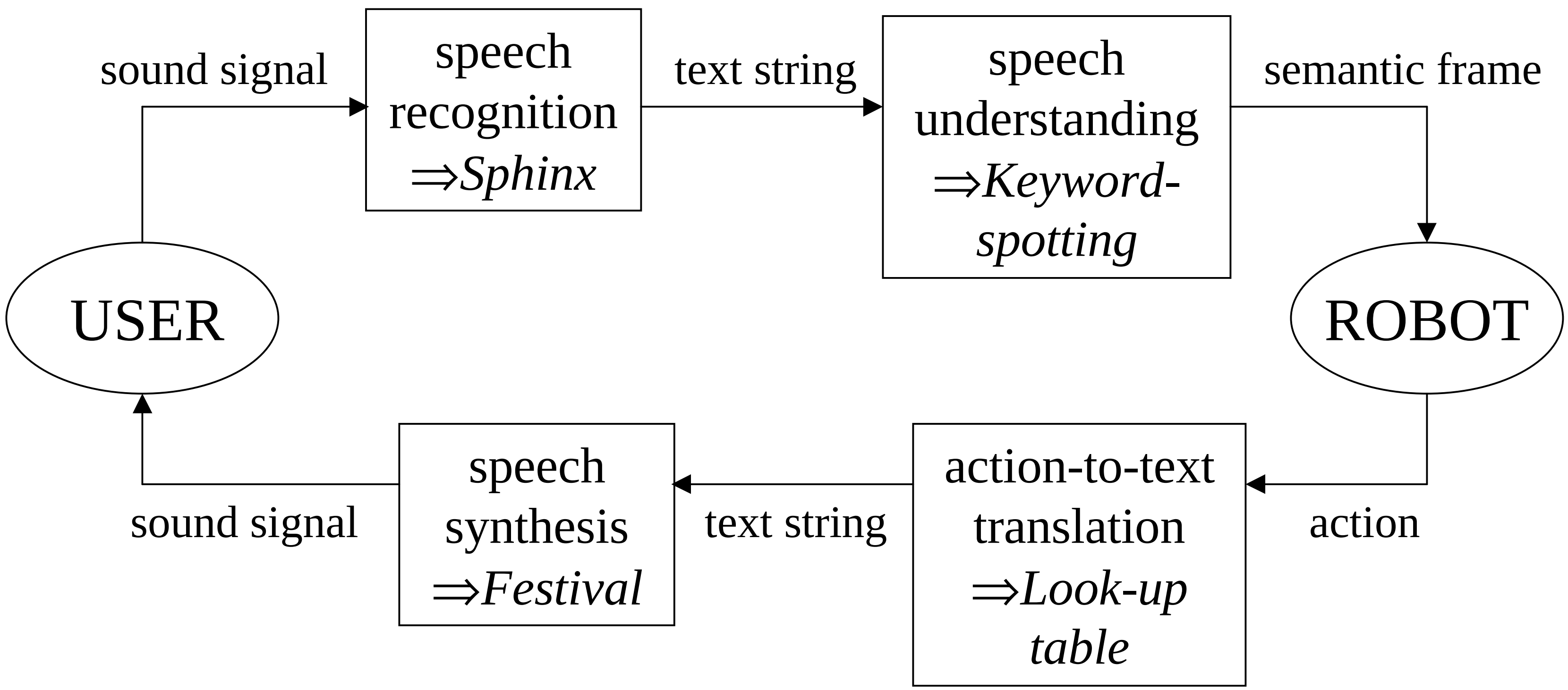
February, 2000

Joelle Pineau

Overview

- The dialogue management problem.
- Previous approaches to dialogue management:
  - Scripts,
  - Information-based task modeling,
  - MDPs.
- Using POMDPs for dialogue management.
- Hierarchical POMDPs.

An Interactive Speech Interface



Dialogue Management

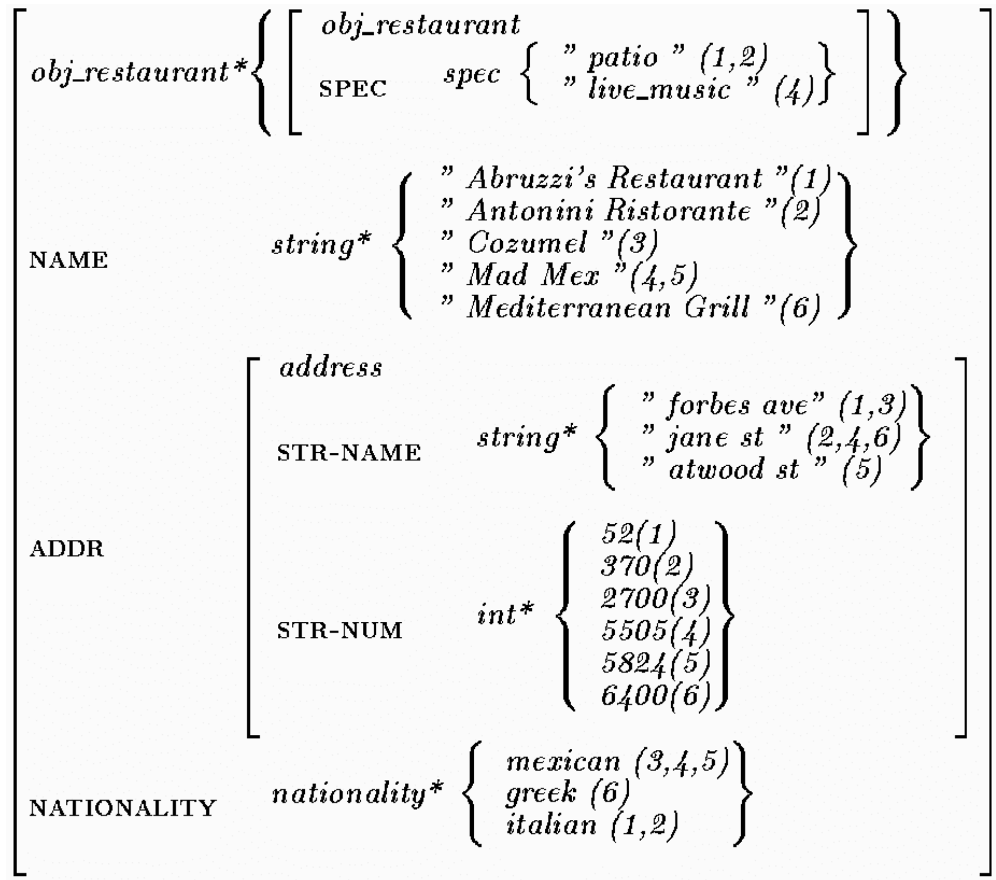
- The goal:
  - Offer set of services satisfying the communicative goals of the user.
- The design process:
  - search the space of possible dialogue strategies for one that is relatively natural without compromising system effectiveness.
- Approaches:
  - Scripts, frames, MDPs, POMDPs

1. The Script Approach

- Use a state machine.
  - State is parameterized by:
    - initial prompt to enter the state;
    - a grammar specifying what the user can say.
  - State transitions are driven by user utterances.
- Dialogue strategy is provided by designer.
- Strategy is optimized through iterative data collection and design refinement.

2. The Information-Based Approach

- Represent objects using *underspecified* typed feature structures.
- Missing information is transformed to disjunctively specified representations.
- Use clarification questions to resolve disjunctions and meet lower bound on information specificity.
- The choice of action is determined by the specificity of available information.



3. The MDP Approach

- Specify a Markov Decision Process:
  - a set of states, *S*,
    - represent the system's state (e.g. entire dialog so far), including current communicative goals and relevant world information;
  - a set of actions, *A*,
    - describe the system's effect on the world, such as responses and information queries;
  - a set of rewards, *R*
  - a set of transition probabilities, $T(s,a,s') = Pr(s' \mid s,a)$.
- Apply reinforcement learning to find a good action-selection strategy.

MDP Example Dialogue Manager:
An automated travel agent conversation system

- Action set: {SayGreeting, AskDepartureCity, AskArrivalCity, AskDepartureDate, AskDepartureTime, AskAll, ConfirmAll}.
- State set: boolean values for {GotDepartureCity, GotArrivalCity, GotDepartureDate, GotDeparture, ConfirmedAll}.
- Reward function: at end of the dialog, user indicated whether they would {use the system again (+1), not use the system again (-1)}.
- View the dialogue as a sequence: $d = (a_1, o_1, r_1), (a_2, o_2, r_2), \ldots, (a_t, o_t, r_t)$

  E.g. (SayGreeting, "I want to go to Pittsburgh",0), (AskDepartureCity, "From Philadelphia",0), …, (ConfirmAll, "yes", ±1)
- Transform the dialogue to a state-space trajectory:

$$s_o \rightarrow_{a1} s(d[1]) \rightarrow_{a2} s(d[2]) \rightarrow_{a3} \ldots \rightarrow_{at} s(d[t])$$

E.g. $[00000] \rightarrow_{\text{SayGreeting}} [01000] \rightarrow_{\text{AskDepartureCity}} [11000] \rightarrow \ldots$

Limitations of MDP Approach

- Requires great amounts of training data to learn the model.
  - In practice: start with a hand-crafted sensible policy and optimize from there.
- Approximates the state of the system, which could render dialogues non-Markovian.
  - In practice: select state approximation to ensure robustness of results.
- Assumes full observability.

4. The POMDP Approach

- Specify a POMDP model:
  - a set of states, *S*,
    - represent user's current communicative goals and relevant world information;
    - no need to represent the dialogue explicitly in the state representation - this is contained in the belief state;
  - a set of actions, *A*,
    - describe the system's effect on the world, such as responses and information queries;
  - a set of observations, *O*,
    - represent the input received, such as user speech utterances;
  - a set of rewards, *R*,
  - a set of initial state probabilities, $P(s) = Pr(s)$;
  - a set of transition probabilities, $T(s,a,s') = Pr(s' \mid s,a)$;
  - a set of observation probabilities, $O(o,s,a) = Pr(o \mid s,a)$.

Evaluating the POMDP Approach

- Benefits:
  - Ability to represent the user's state directly, without maintaining the dialog history in the state representation.
  - Ability to handle partial observations.
- Difficulties:
  - Having to specify the entire model, especially the probabilistic information.
  - Learning an optimal policy for a non-trivial model.

Hierarchical POMDP Approach

- Divide the domain into a hierarchy of related subtasks.
- Each subtask is defined as a separate POMDP model.
- To create a subtask $M'=(S',A',O)$, subordinate to subtask $M=(S,A,O)$:
  - remove the subset S' from S,
  - remove the subset A' from A,
  - add $s_k=child_k$ to S, where $child_k$ is the sub-domain M',
  - add $a_k=use\text{-}child_k$ to A, where $use\text{-}child_k$ corresponds to selecting to use sub-model M',
  - use the same observation set, O, for both models.
- Condition: $\forall a_k \in A'$: $\Pr(o|s_i,a_k) \neq \Pr(o|s_j,a_k)$ or $R(o,s_i,a_k) \neq R(o,s_j,a_k)$, for some $o \in O$ and for some $(s_i,s_j) \in S'$.

Flo's Conversational Domain

top-POMDP

States: {none, TV-domain, weather-domain, calendar}
Actions: {initiate, use-tv, use-weather, say-calendar, specify-schedule}
Observations: {none, noise, abc, cbs, nbc, tv, now, later, weather, current, today, tomorrow, see, hear, schedule, calendar}

tv-POMDP

States: {abc-now, abc-later, cbs-now, nbc-now}
Actions: {default, ask_time, ask_channel, say_abc-now, say_abc-later, say_cbs, say_nbc}
Observations: {none, noise, abc, cbs, nbc, tv, now, later, weather, current, today, tomorrow, see, hear, schedule, calendar}

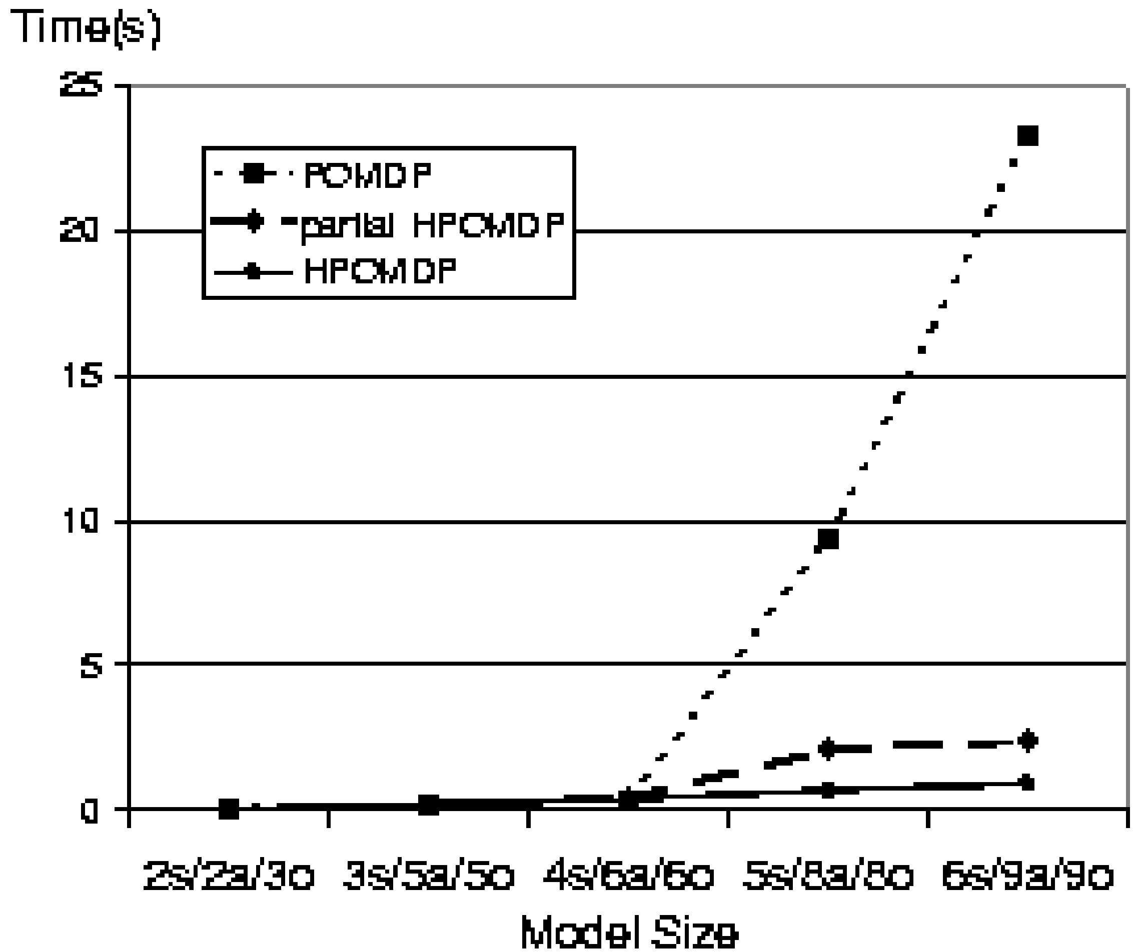
weather-POMDP

States: {current-weather, today-weather, tomorrow-weather}
Actions: {default, ask_time, say_current, say_today, say_tomorrow}
Observations: {none, noise, abc, cbs, nbc, tv, now, later, weather, current, today, tomorrow, see, hear, schedule, calendar}

Sample Conversation

R: "How can I help you?"	$Act_{top-POMDP}$=Initialization
U: "What is playing on TV?"	*Observation*=tv
R: "On what channel?"	$Act_{top-POMDP}$=Use-tv $Act_{tv-POMDP}$=Select-channel
U: "On ABC."	*Observation*=abc
R: "For what time?"	$Act_{top-POMDP}$=Use-tv $Act_{tv-POMDP}$=Select-time
U: "Tell me the weather instead."	*Obervation*=weather
R: "For which day?"	$Act_{top-POMDP}$=Use-weather $Act_{weather-POMDP}$=Select-day
U: "How about today?"	*Observation*=today
R: "The forcast is rain."	$Act_{top-POMDP}$=Use-weather $Act_{weather-POMDP}$=Say-today
U: "Thanks, so about the TV..."	*Observation*=tv
R: "Yes, for what time?"	$Act_{top-POMDP}$=Use-tv $Act_{tv-POMDP}$=Select-time
U: "Now."	*Observation*=now
R: "ABC is now playing News at 6".	$Act_{top-POMDP}$=Use-tv $Act_{tv-POMDP}$=Say-abc-now
U: "Thanks."	*Observation*=(none)

Preliminary Results



What happens when we do this?

- What have we gained?
  - now have a collection of smaller models;
  - each model can be solved and designed independently;
  - it is easier to solve all submodels separately than solving the complete flat-POMDP;
  - it is easier to design many smaller models than a single big one.
- What have we lost?
  - can’t guarantee global optimality of solution anymore.
- Similar to MAX-Q decomposition:
  - collection of related specialized subtasks, each designed and solved independently,
  - but no need to specify conditions to exit a particular subtask.

Remaining questions:

- Do we really need to maintain all observations in all models?
  - No, but how can we specify *local* vs *global* observations?
- How can we facilitate the design of the probabilistic distributions?
  - Is there an easy way to specify probabilities?
  - Is there an effective way of learning or at least updating probabilities?
- How can we perform decomposition automatically?
- How can we select decomposition that optimizes trade-off between:
  - maximal model reduction to achieve faster learning, and
  - minimal policy space constraints to improve optimality?
- Have we achieved *hierarchical optimality* or *recursive optimality*?
- Can we achieve value function decomposition?