Parallel Computing and Practical Constraints when applying the Standard POMDP Belief Update Formalism to Spoken Dialogue Management

Paul A. Crook, Brieuc Roblin, Hans-Wolfgang Loidl and Oliver Lemon

Abstract We explore the commonly stated assumption that the *standard POMDP formalism* for belief updates cannot be directly applied to Dialogue Management for Spoken Dialogue Systems (SDSs) due to the computational intractability of maintaining a large belief state space. Focusing on SDSs, as this application has particular bounds in terms of "real-time" belief updates and potentially massive numbers of observations, we quantify computational constraints both in terms of compute time and memory. We establish a level of complexity of SDS task below which a direct implementation of the standard POMDP formalism is possible and beyond which some form of compressed representation is required. We find that computation time of POMDP belief updates is rarely an issue. Memory size and latency tend to be the dominant constraints. Low-latency, shared-memory architectures are more suitable than General Purpose Graphics Processing Units (GPGPUs) or large-scale cluster/cloud infrastructure. One assumption, that users do not change their goal during a dialogue, has significant beneficial impacts on memory requirements allowing for practical POMDP SDSs which have millions of states.

1 Introduction

Modelling dialogues as Partially Observable Markov Decision Processes (POMDPs) has been proposed to improve robustness and reduce error rates for Dialogue Management of Spoken Dialogue Systems (SDSs) [15, 7, 18, 13].

The *standard POMDP formalism*, by which we mean an implementation of a POMDP where all the possible states of the task are *fully* enumerated as part of the design and *remain fixed* through out the duration of the task, *i.e.* there is no approximation, compression, recombination or pruning of states, is typically assumed to be an impractical approach to Dialogue Management of SDSs due to the "astronomical number of possible user goals, and so computing this update directly is impossible in real time" [16]. Similar views on tractability have been expressed by other authors [13, 18]. To this end alternative approaches have been proposed which more compactly represent the states of the dialogue. These approaches include grouping states into partitions [18], maintaining parallel dialogue paths [7], and Bayesian network

Paul A. Crook, Brieuc Roblin, Hans-Wolfgang Loidl, Oliver Lemon
School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK
e-mail: {p.a.crook, h.w.loidl, o.lemon}@hw.ac.uk, brieuc.roblin@gmail.com

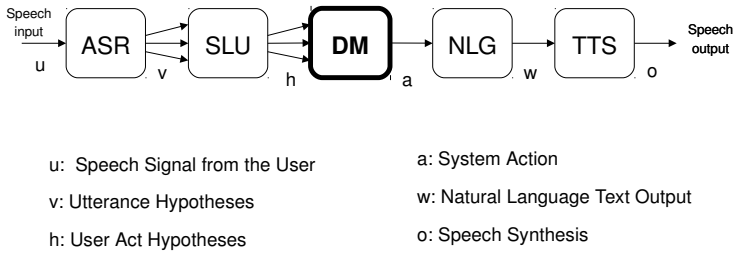


Fig. 1 Outline of a typical SDS. Dialogue Manager is highlighted.

models [13]. However these approaches do have some limitations. The trade-off is typically either loss in accuracy or imposition of additional structure (independence assumptions between features) which limits the dialogue states that can be represented. Some approaches also require an additional recombination or pruning step as part of the belief update [7, 18] which itself can be computationally expensive [6]. Such a step is not required by the standard POMDP formalism.

The aim of this paper is to examine the above commonly stated assumption in light of the increasing availability of parallel computation and quantify the computational limitations both in terms of compute time and memory. To our knowledge practical limits for the application of the standard POMDP formalism to SDS tasks have never been quantified. We consider the domain of SDSs engaged in limited-domain query-dialogues (a.k.a. "slot filling" tasks). Note that our focus is on the *belief update* step. Learning is not considered in this paper. We make the assumption that either off-line approximate POMDP solution techniques, *e.g.* Perseus [12] or, *more likely*, sample efficient approaches to Reinforcement Learning [3] can be used to learn a policy which maps belief states to actions.

As well as *quantifying computation bounds* for a SDS implemented using the standard POMDP formalism, the *other* key contributions of this paper are: *(i)* Showing that the commonly adopted assumption that the user goal does not change during the course of a dialogue is extremely beneficial in terms of *reducing the memory requirements* of the belief update. *(ii)* Making the observation that using *sets of goals* as the user's goal state [4] partially mitigates the restrictiveness of the aforementioned unchanging user goal assumption.

1.1 Paper Structure

The remainder of this paper is organised as follows. Section 2 outlines the typical SDS architecture, introduces the standard POMDP formalism and details the POMDP *belief update* step. Section 3 reviews related literature including the establishment of an upper bound on the time available for SDS Dialogue Manager (DM) belief updates. Section 5 sets out the methodology and equipment used. Section 6 reports the limits found for belief updates given a *generic* POMDP task. Section 7 reports limits for belief updates tailored more specifically to SDS DM and outlines practical SDSs that could be built within these constraints. Finally Section 8 concludes.

2 Background

2.1 Typical SDS

A typical SDS follows the basic outline as shown in Figure 1, though most current systems do not consider multiple hypotheses when processing user input. It consists of an Automatic Speech Recogniser (ASR) which maps sound signals received from the user to word strings, a Spoken Language Understanding (SLU) unit which then maps the word strings to semantic concepts, and a DM which receives the semantic version of the user's utterance, monitors and updates the current dialogue state, and decides what action to take next. The Natural Language Generator (NLG) then maps the DM's action into natural language text which is then realised as spoken speech by the Text To Speech (TTS) unit. This paper focuses on one part of the process carried out by the DM, that of updating its view of the dialogue.

We specifically consider POMDP based DMs. That is the current dialogue is modelled by the DM as a Partially Observable Markov Decision Process.

2.2 POMDP Formalism

A Markov Decision Process (MDP) typically describes some system interacting with an environment where the outcomes achieved by the system are partly random (due to stochasticity in the environment) and partly under the control of the system through the actions it decides to take. The distinguishing feature of a MDP is that it has the *Markov property*. That is the property that the current state of the environment (as observed by the system) always contains sufficient information to make an optimal decision about what action to perform next (*e.g.* all relevant history is represented in the observed state). A POMDP is a generalisation of a MDP where the system cannot directly observe the state of the environment. Instead it maintains a probability distribution, commonly referred to as its *belief*, over the set of possible states that the environment could be in. The system's belief is updated by combining the observations it receives as it acts in the environment, its knowledge of the observation probabilities and its knowledge of the transition probabilities in the underlying MDP. Formally, a POMDP is defined as a tuple $\langle S, A, O, T, \Omega, \mathscr{R} \rangle$ where S is the set of states that the environment can be in, A is the set of actions that the system can take, O is the set of observations which it can receive, T is a set of conditional transition probabilities which describes the likelihood of transitioning between states given a selected action, Ω is a set of conditional observation probabilities which describes the likelihood of each observation occurring and $\mathscr{R}$ is the reward function ($\mathscr{R} : A, S \rightarrow \mathbb{R}$).

Equation 1 is the typical form of a POMDP belief update equation. b is the system's *belief* at the current time; it is the probability distribution of state occupancy over all possible states of the environment S. The probability of being in any one state $s \in S$ given the current belief b is denoted as $b(s)$ where $b(s) = P(s \mid b)$. Assuming action a is taken and o is observed then $b'(s')$ is the belief that the agent is in state s' at next time step given it observes $o \in O$ with probability $\Omega(o \mid s', a)$.

$T(s' \mid s,a)$ is the probability of transitioning from state s to state s' when action a is taken. η is a normalising factor.

$$b'(s') = \eta\ \Omega(o \mid s',a) \sum_{s \in S} T(s' \mid s,a) b(s) \tag{1}$$

We can represent the belief b as an $|S|$ dimensional column vector $\mathbf{b}$ with elements $\mathbf{b}_i = b(s_i)$. Define an *observation vector* $\Omega^{\mathbf{o},\mathbf{a}}$ as an $|S|$ dimensional column vector with elements $\Omega_i^{\mathbf{o},\mathbf{a}} = \Omega(o \mid s_i,a)$, a *transition matrix* $\mathbf{T}^{\mathbf{a}}$ as an $|S| \times |S|$ dimensional matrix whose elements are $\mathbf{T}^{\mathbf{a}}{}_{i,j} = T(s_i \mid s_j,a)$, $\circ$ as the element-wise product between the elements of two vectors (Hadamard product) and $\cdot$ as the conventional dot product. Equation 1 can then be re-written as:

$$\mathbf{b}' = \eta\ \Omega^{\mathbf{o},\mathbf{a}} \circ (\mathbf{T}^{\mathbf{a}} \cdot \mathbf{b}) \tag{2}$$

This re-write is useful as we can then think of the update in terms of the need to store, retrieve and carry out multiplication operations between vectors and matrices of dimension $|S|$ and $|S| \times |S|$ respectively. We can also see that for this standard formulation of the POMDP problem we will need to store $|A|$ transition matrices (one for each possible action a) and $|A| \times |O|$ observation vectors (one per each possible action a and observation o).

3 Related Literature

We found little literature that considered the computational bounds of POMDP belief updates. Williams, Poupart and Young (2005) [14] and Williams and Young (2005) [17] implement a SDS using what we have termed the standard POMDP formalism. Although they are concerned with scaling up to large SDSs they do not quantify what size belief updates can be computed. Most POMDP literature focuses on the computation bounds encountered when trying to compute optimal POMDP policies, *e.g.* Spaan and Vlassis (2004) [12], and not the belief update step required to execute the learnt policies. The most specific results for POMDP SDS belief update are those presented by Gašić and Young (2011) [6], for the HIS DM, which indicate a roughly linear increase in update time with increasing numbers of partitions until approximately 3,800 partitions. The update time then leaps dramatically from 381 ms for approximately 3,800 partitions to 1.138 seconds for approximately 4600 partitions.[1] This jump is presumably due to the update exceeding some memory constraint and thus requiring the machine to utilise additional, slower storage.

3.1 Dialogue Response Time

A requirement specific to SDSs is that the system should respond in "real-time" to human queries. In the evaluation of SDSs researchers often record the inter-utterance delay between the user finishing speaking and the system response [10]. However as there is rarely a strong correlation between this and user satisfaction these figures are rarely reported.

[1] The machine used had 24 GBytes of memory and an 8 core Intel Xeon 2.83 GHz CPU.

Some work looking at the effect of delays on user's subjective assessment of quality has been done for telecommunications services as summarised in Möller (2000) [9], where contrasting results indicate that experts perceive a reduction in quality for a delay exceeding 200 ms, but subjective ratings in user trials were only marginally effected by adding a delay of 600 ms.

Often when constructing SDSs researchers have taken a pragmatic view of response times, *e.g.* "In order for the system to run in real time the total response time (including decoding and generation) should not be larger then 1 second, which means that the belief update time should be kept well below that threshold" [6].

Perhaps the best guide to what should be the response time for an SDS is given by human-human dialogue corpora, *e.g.* Raux (2008) [11] (pp.40–45) and Bull and Aylett (1998) [2]. The former considers a limited-domain query-dialogue task of providing bus routing and timetable information to the general public [1].[2] In human-human dialogues where an *operator* is responding to queries over the phone from the general public, the average operators' response delay[3] was 700 ms. A histogram of the response delays indicates that this mean is not drawn from a normal distribution as over 77% of the pauses lie below 700 ms and there is a very long flat tail out to 3000 ms. These results are explored further by classifying the response delays into three categories according to the utterance that they precede [11]; "initiation" where a new expectation (*e.g.* a new goal or question) is added to the dialogue, "response" the fulfilment of an expectation (*e.g.* acknowledgement, response to yes-no or wh- question), "closing" end of dialogue moves. This breakdown indicates that initiation moves are preceded by an average delay of 764 ms, response moves by an average delay of 361 ms and closing moves by an 82 ms average delay.

Bull and Aylett (1998) [2] looked at the timing of turn taking in an instruction giving-receiving map task. They report the variation in inter-speaker intervals for a variety of conditions. For all conditions the typical mean response delay lies between 400 and 600 ms (with the exception of game boundaries where the mean delay is 621 ms). Possibly of the most relevance is that they found that participants seemed compelled to respond more quickly when they were not able to see each other; mean response delay of 422 ms compared to 579 ms for the condition when eye contact was possible. This evidence suggests that a response time of around 400 ms is desirable in order to match human expectations. Thus we conclude that the POMDP belief update, which is just part of the processing carried out by the SDS, should take no more than 400 ms.

4 POMDP DM

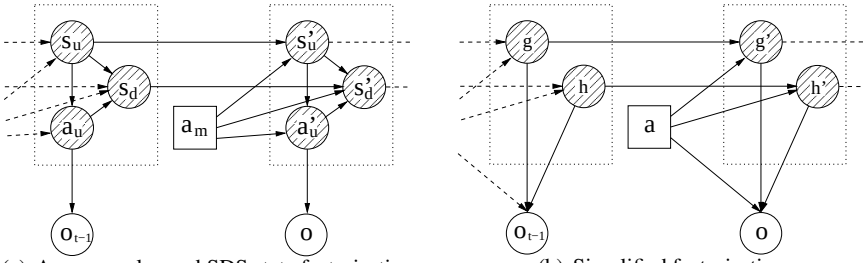
For POMDP DMs we say that the user's utterance after it has been rendered into the form of a semantic act (or a list of semantic acts[4]) is the observation o which the POMDP receives. We assume that the dialogue has a discrete number of states which it can be in and these are represented by the set of POMDP states S. Finally

[2] This is the "Lets Go" task on which SDSs have been employed.

[3] Measured as "a period of joint silence bounded by vocalisations of different speakers."

[4] In the case of a system that considers N-best lists of ASR output.

the DM's action is equated to the POMDP act a. Now given a set of transition matrices, observations vectors and an initial belief b_0 the POMDP DM can monitor and update its belief b over the possible states of the dialogue. We say that this is a *generic POMDP task* as we do not suggest exactly what information the states $s \in S$ should contain and thus we are *not* exploiting any known structure of the task to simplify the computation. Thus this *generic POMDP task* is a worst case scenario.



(a) A commonly used SDS state factorisation (b) Simplified factorisation

Fig. 2 Hatched nodes indicate hidden variables. Left hand side: Williams and Young (2007) [15] s_u is the user's goal, s_d is the dialogue history, a_u estimated "true" user action and a_m the SDS action. Right hand side: g is the user's goal, h is the dialogue history and a the SDS action.

We also consider a state representation s that is more specifically tailored to SDS and we exploit an assumption that other authors, *e.g.* Williams and Young (2007) [15], have made in this domain, that users do not change their goal during the course of a dialogue, in order to reduce the computational load. We consider that the state s can be factored into two parts; the user's goal g and some features which summarise the dialogue history h. Where G is all the possible user goals and H is the possible histories the system can represent. This factorisation is similar to the factored SDS-POMDP state proposed by Williams and Young (2007) [15] but leaves out a hidden variable which estimated the previous user action. Instead both the user goal g and history h are directly dependent on the observation o, and the likelihood of the observation is dependent on the machine action a; *c.f.* Figures 2(a) and 2(b). We prefer the factorisation shown in 2(b) as it results in a more compact state space with only a small loss in generality. Using this factorisation Equation 1 becomes:

$$b'(g',h') = \eta\ \Omega(o \mid g',h',a) \sum_{g \in G} \sum_{h \in H} T(g',h' \mid g,h,a) b(g,h) \tag{3}$$

If we make the not unreasonable assumption (for a limited-domain query-dialogue) that the likelihood of the dialogue history is independent of the user's goal the factorised form can be simplified:

$$b'(g',h') = \eta\ \Omega(o \mid g',h',a) \sum_{g \in G} \sum_{h \in H} T(g' \mid g) T(h' \mid h,a) b(g,h) \tag{4}$$

Finally we incorporate the assumption that users do not change their goal during the course of a dialogue, see Section 4.1 for discussion. This results in term $T(g' \mid g)$ becoming a Kronecker delta function which has the value 1.0 only when $g = g'$. The

summation over g thus disappears leaving us with Equation 5.

$$b'(g',h') = \eta\ \Omega(o \mid g',h',a) \sum_{h \in H} T(h' \mid h,a) b(g',h) \tag{5}$$

In matrix/vector notation this factorisation and assumptions reduce the term $\mathbf{T^a}$ in Equation 2 to a block diagonal matrix with repeated copies of $\mathbf{H^a}$ on the diagonal, *i.e.*

$$\mathbf{T^a} = \begin{bmatrix} \mathbf{H^a} & 0 & \cdots & 0 \\ 0 & \mathbf{H^a} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{H^a} \end{bmatrix}$$

where $\mathbf{H^a}$ is a square matrix with elements $\mathbf{H^a}_{i,j} = T(h_i \mid h_j, a)$. This reduces both the amount of computation and the amount of memory required as, assuming the number of possible goals outweighs the number of features used to summarise the dialogue history, $\mathbf{T^a}$ now contains mostly zeros. Further we only need to store one copy of the matrix $\mathbf{H^a}$ (of dimensions $|H| \times |H|$) for each action a rather than the full matrix $\mathbf{T^a}$ (of size $|S| \times |S|$) for each action.

4.1 Fixed Users' Goals during Dialogues

The assumption that users' goals do not change during the course of a dialogue is not as strict as it sounds as it is only a specification of the state transition dynamics model and provided there is some recognition uncertainty all goals will remain reachable, though retraction may require repeated attempts by the user in order for the system to accumulate evidence of the change of goal.

For tasks such as accessing contact details from a phone directory or booking flights this assumption probably holds. For other domains such as a restaurant guide [8], where the user may well be browsing, it could lead to difficulties. One approach that mitigates the effects of this assumption is the suggestion by Crook and Lemon (2010) [4] to represent user goals not as singular items but as *sets* of items. Using this approach a possible user goal could be, for example, {`French restaurants nearby \/ Chinese near home \/ ...` }. Using this formulation a user who changed their mind would, unless they persistently negated a previous statement, be seen as browsing a set of options.

Of course this approach significantly expands the state space of possible user goals with the set of goal sets being equal to 2 to the power of the number of possible combinations, *i.e.* given a restaurant task with $|food|$ possible foods and $|areas|$ possible city areas the set of goal sets is $2^{|food| \times |areas|}$.

5 Methodology

We investigated the belief update time of the standard POMDP formulation by implementing Equation 2. Three flavours of the algorithm were developed; a serial C implementation which runs on one core of a CPU, a parallel, shared-memory C implementation using the OpenMP pragmas to distribute processing across multiple

CPU cores, a CUDA implementation which uses the massively parallel computational abilities of General Purpose Graphics Processing Units (GPGPUs). These algorithms were run with increasing number of states $|S|$ until host memory limitations were reached. Double precision floating point values were used in all cases.

Two scenarios were considered. The first used completely generic transition matrices and observation vectors, *i.e.* not tailored to a particular task. The second consider transition matrices which are more specifically tailored to a particular SDS problem see Section 4. To give a worst case scenario matrices and vectors where assumed to be *dense* (as opposed to *sparse*, *i.e.* containing mostly zeros).

Using the standard POMDP formalism the number of states is permanently fixed by the system designer (*c.f.* partition based POMDP SDSs where the number of partitions tends to grows with time). Fixing the number of states implies that the belief update time is more-or-less constant. To examine the variation of belief update times with different numbers of chosen states we repeatedly ran systems with increasing number of states and sampled belief update times for 1,000 consecutive updates. The observation and transition matrices used were populated with random entries whilst ensuring that they were properly formed probability distributions.

Memory constraints were established by considering the size of vectors and matrices that need to be stored and simply computing the maximum possible given the machine's available physical RAM (excluding any swap space).

Serial and OpenMP tests were carried out using an eight core CPU machine (two 4 core Intel Xeon E5506 processors) running at 2.13 GHz. CUDA tests used an NVIDIA TESLA GPGPU card, model C2050, which has 448 CUDA cores and 3 GB of memory, the host machine is a Dell Precision T5500 with a 4 core Intel Xeon X5550 2.67 GHz CPU and 12 GBytes of memory.

6 Dense POMDP Belief Updates

Figure 3 shows average POMDP belief update times versus number of states $|S|$. The serial version which utilises only a single core of the CPU is, as expected, the slowest. For low numbers of states the GPGPU version is clearly the quickest but once it is no longer possible to store the complete set of dense transition matrices $\mathbf{T^a}$ in the 3 GBytes of GPGPU memory then bus transfer times between the host and GPGPU card dominate the update slowing it considerable. We assume a worst case condition of having to load a transition matrix each time. If actions were commonly repeated then caching of transition matrices would improve the average GPGPU update time for more than 4,400 states.

The OpenMP implementation which uses all eight cores of the CPU out performs the GPGPU for high numbers of states, taking less than 400 ms to compute the belief update for up to 26,000 states. However memory is still an issue as a dense transition matrix of $|S| \times |S|$ is required for *each* action and assuming, say, 20 actions then 8,192 states is the maximum that can be accommodated in 10 GBytes of memory[5] (for a POMDP task with only 10 actions the maximum number of states would be approximately 11,500). Beyond this point the transition matrices would have to be

[5] The target machine has 12 GBytes of memory but some overhead is assumed for other processes.

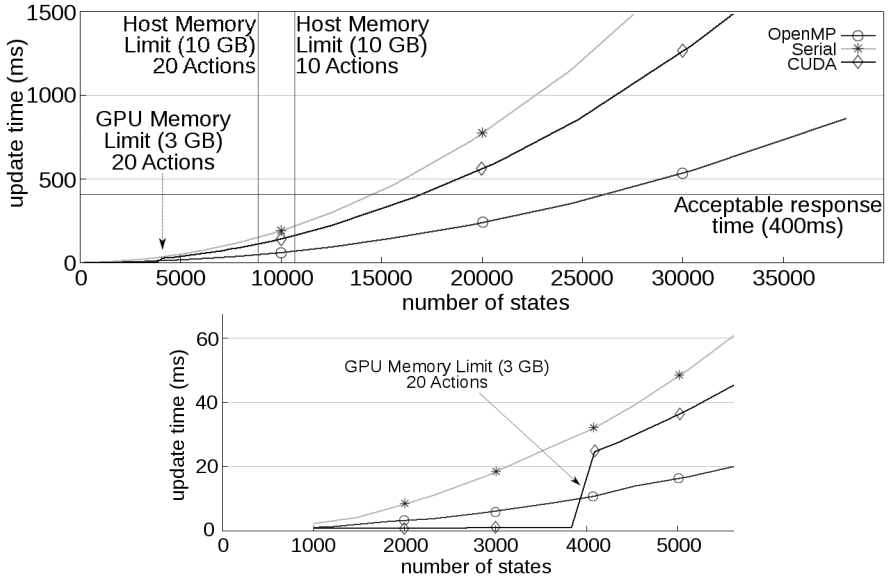


Fig. 3 Plot of average POMDP belief update time versus number of states $|S|$ for dense matrices. Lower plot shows a close-up of the initial 5,000 states.

retrieved from the hard disk drive (HDD). Currently the top sustained transfer rate for HDD is around 128 MBytes/s, thus it would take 4 seconds to retrieve an 8,192 $\times$ 8,192 double precision matrix making this option impractical.

The main limitations for dense POMDP belief updates can be summarised as follows for 8 byte double precision[6] floating point values:

- $|S|^2 \times |A| \times 8$ Bytes $\leq M$ where M is the machine's memory.
- $\frac{4\times|S|^2+2\times|S|}{(1+(C-1)\times\sigma)\times FLOPS} < 400$ ms where $FLOPS$ is floating point operations per second for *single precision numbers* on *one core*, C is the number of cores and σ is the speed up from additional cores.

In our case we achieve around 1.15 GFLOPS per core and each additional core contributes an approximate speed up of $\sigma = 0.6875$, see Figure 4.[7]

7 Limits for SDS DM

Using the assumptions set out in the latter part of Section 4 to simplify the update, the size of the transition matrices ceases to be an issue as the memory constraint becomes of $|H|^2 \times |A| \times 8$ Bytes $\leq M$ and typically for limited-domain querydialogues four tri-valued features could be used to summarise the history resulting in $|H| = 3^4 = 81$ states. The compute time is also reduced dramatically to $\frac{4\times|H|^2+2\times|G|+6\times|H||G|}{(1+(C-1)\times\sigma)\times FLOPS}$. To arrive at this formula we presume the computation of

[6] For 4 byte single precision floats the compute times and memory requirements can be halved.

[7] This is not using any Streaming SIMD (Single Instruction Multiple Data) Extensions (SSE).

marginals for $b(g)$ and $b(h)$ such that the matrix multiplication involving $\mathbf{H^a}$ only occurs once. As an example if $|H| = 81$ and the number of user goals $|G| = 15,000$ this then results in 1,215,000 states, $|S| = |H||G|$, but an estimated compute time of less than 1 ms on our test machine using the OpenMP implementation.

The main constraint is the number of observation vectors that have to be stored. Depending on the flexibility of the grammar/semantics of the ASR/SLU there can be thousands (possibly millions) of observations even in a limited domain. For each observation there are $|A|$ *dense* vectors of size $|S|$ containing double precision floating point numbers, thus HDDs are currently the only practical storage solution. Storing observation vectors on a modern HDD should not result in the update time exceeding our target of 400 ms until $|S| > 6{,}000{,}000$.[8] However systems are constrained by the total storage space required on the HDD which is $|O| \times |A| \times |S| \times 8\,\text{Bytes} < HDD\_storage$.

7.1 Practical Systems

Given a target machine with a 4 core Xeon 2.67 GHz CPU, 12 GB RAM and 160 GB hard drive, Table 1 gives two examples of practical POMDP DMs that could be constructed.[9] The HDD capacity is the major limitation, for example it restricts the possible semantics that the restaurant guide system could understand.

Task	Company Phone Directory 300 employees, 3 cities	City Restaurant Guide 130 restaurants in central area of Edinburgh
Observations	120,000 (300 × surname, 100 × first name, 4 × city (includes null))	$6 \times \|food+1\| \times \|areas+1\|$ based on semantics of the form: `[yes|no] [negation] [food][area]` where each term can be omitted (factor of 6 from possible combinations of yes/no and negation).
States	$\|H\| = 3^3$, $\|G\| = 300$ $\Rightarrow \quad \|S\| = 8,100$	User goal represented by *sets* over possible restaurant combinations, see Section 4.1 $\|G\| = 2^{\|food\|\times\|area\|}$, $\|H\| = 3^2$ $\Rightarrow \quad \|S\| = 9 \times 2^{\|food\|\times\|area\|}$
Actions	$\|A\|$ = 20	$\|A\|$ = 20
HDD storage	**156 GBytes**	$\|food\|$=9 & $\|areas\|$=2 $\Rightarrow$ 180 observations; $\|S\|$=2,359,296; **68 Gbytes** $\|food\|$=6 & $\|areas\|$=3 $\Rightarrow$ 168 observations; $\|S\|$=2,359,296; **63 Gbytes**
Update time	estim. 5 ms	estim. 155 ms

Table 1 Two practical POMDP systems within the constraints of the target hardware, using the simplifying assumptions for SDS limited-domain query-dialogue tasks as set out in Section 4.

8 Conclusions

We establish a level of complexity of SDS task below which a direct implementation of the standard POMDP formalism is possible and beyond which some form of

[8] Based on a HDD seek time of 4.2 ms and sustained reading rate of 126 MBytes/s.

[9] ASR, SLU, NLG and TTS processing are assumed to take place on separate servers.

compressed or approximate representation is required. This serves to establish an important baseline for future POMDP SDS work.

Given the increasing parallelism of modern processors the time to compute belief updates does not appear to be an issue in using the standard POMDP formulation for SDS Dialogue Management. Further, the central computation of the belief update is highly parallel. Tests with the OpenMP implementation indicate that there is an almost linear speed-up in belief update compute time as more CPU cores are made available, see Figure 4, and the speed-up is insensitive to the size of the state space.

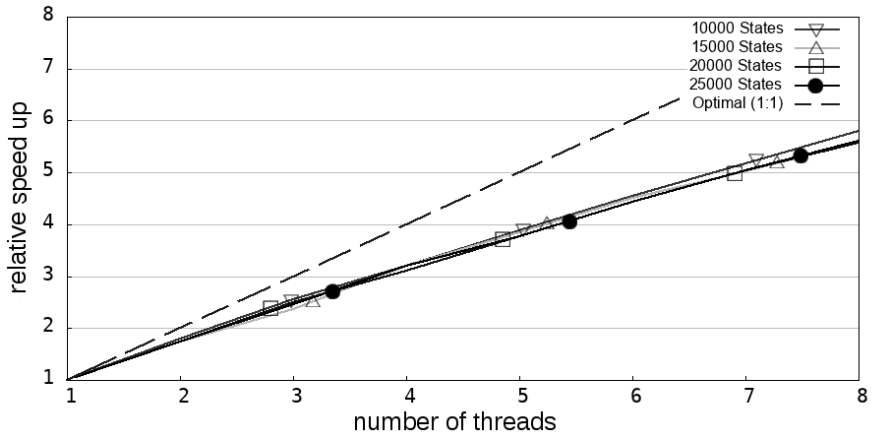


Fig. 4 Plot of speed-up of POMDP belief update versus number of CPU cores used for execution.

SSE[10] instructions provide an orthogonal speed-up to the OpenMP approach. When used in combination we found a further 1.5 fold speed-up.[11]

CPU cores suffer very little overhead in terms of data transfer due to the shared-memory nature of the PC architecture. In contrast data transfer proved to be critical to the CUDA implementation which uses a deep memory hierarchy and exhibits a very high memory latency. The GPGPU was limited to smaller *generic POMDP tasks* by its 3 GB memory. The use of a card with more memory or multiple cards within the same machine would boost the size of problems that can be computed by allowing large transition matrices to be cached in the memory of multiple cards.

We conclude that the main limitation for this form of POMDP implementation is imposed by memory for storage of transition matrices and observation vectors and the resulting time taken to retrieve these stored matrices and vectors. Thus, a low-latency, shared-memory architecture is most suitable despite its inherent limits on the degree of parallelism. This precludes any benefit in moving to large-scale distributed architectures such as compute clusters or cloud infrastructure.

If the POMDP task is more specifically tailored to SDS Dialogue Management, for example the factorisation and assumptions set out in Section 4, large numbers

[10] Streaming SIMD (Single Instruction, Multiple Data) Extension.

[11] A 4 fold speed-up was expected but compiler optimisations meant the baseline implementation was more efficient than anticipated.

of states are much more manageable and practical POMDP SDSs with millions of states can be constructed using current, non-specialist hardware.

Future work includes: *(i)* Building and testing a real restaurant guide SDS along the lines indicated. *(ii)* Research on compression techniques for exploiting state and observation structure to automate the compressions required for larger SDSs [5].

Acknowledgements The authors would like to thank the Engineering and Physical Sciences Research Council, UK (EPSRC) grant number EP/G069840/1, and partial funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270019 (SPACEBOOK project www.spacebook-project.eu).

References

1. Black, A.W., Burger, S., Conkie, A., Hastie, H., Keizer, S., Lemon, O., Merigaud, N., Parent, G., Schubiner, G., Thomson, B., Williams, J.D., Yu, K., Young, S., Eskenazi, M.: Spoken Dialog Challenge 2010: Comparison of Live and Control Test Results. In: Prc. SIGdial (2011)
2. Bull, M., Aylett, M.: An analysis of the timing of turn-taking in a corpus of goal-oriented dialogue. In: Proceeding of ISCLP (1998)
3. Chandramohan, S., Geist, M., Pietquin, O.: Sparse approximate dynamic programming for dialog management. In: Proceedings of SIGdial (2010)
4. Crook, P.A., Lemon, O.: Representing uncertainty about complex user goals in statistical dialogue systems. In: Proceedings of SIGdial (2010)
5. Crook, P.A., Lemon, O.: Lossless Value Directed Compression of Complex User Goal States for Statistical Spoken Dialogue Systems. In: Proceedings of Interspeech (2011)
6. Gašić, M., Young, S.: Effective Handling of Dialogue State in the Hidden Information State POMDP-based Dialogue Manager. ACM Transactions in Speech and Language Processing **7**(3) (2011)
7. Henderson, J., Lemon, O.: Mixture Model POMDPs for Efficient Handling of Uncertainty in Dialogue Management. In: Proceedings of ACL (2008)
8. Lemon, O., Georgila, K., Henderson, J.: Evaluating Effectiveness and Portability of Reinforcement Learned Dialogue Strategies with real users: the TALK TownInfo Evaluation. In: IEEE/ACL Spoken Language Technology, pp. 178–181 (2006)
9. Möller, S.: Assessment and Prediction of Speech Quality in Telecommunications. Kluwer (2000)
10. Möller, S., Engelbrecht, K.P., Schleicher, R.: Predicting the quality and usability of spoken dialogue services. Speech Commincation **50**, 730–744 (2008)
11. Raux, A.: Flexible turn-taking for spoken dialog systems. Ph.D. thesis, CMU (2008)
12. Spaan, M., Vlassis, N.: Perseus: randomized point-based value iteration for POMDPs. Tech. Rep. IAS-UVA-04-02, Universiteit van Amsterdam (2004)
13. Thomson, B., Young, S.: Bayesian update of dialogue state: a POMDP framework for spoken dialogue systems. Computer Speech and Language **24**(4), 562–588 (2010)
14. Williams, J., Poupart, P., Young, S.: Factored partially observable markov decision processes for dialogue management. In: Workshop on Knowledge and Reasoning in Practical Dialog Systems (IJCAI) (2005)
15. Williams, J., Young, S.: Partially Observable Markov Decision Processes for Spoken Dialog Systems. Computer Speech and Language **21**(2), 231–422 (2007)
16. Williams, J.D.: Incremental partition recombination for efficient tracking of multiple dialog states. In: Proceeding of ICASSP, pp. 5382 – 5385 (2010)
17. Williams, J.D., Young, S.: Scaling Up POMDPs for Dialog Management: The "Summary POMDP" Method. In: Proceedings of ASRU (2005)
18. Young, S., Gašić, M., Keizer, S., Mairesse, F., Thomson, B., Yu, K.: The Hidden Information State model: a practical framework for POMDP based spoken dialogue management. Computer Speech and Language **24**(2), 150–174 (2010)