Bounded Practical Social Reasoning in the ESB Framework

Iain Wallace
School of Informatics
The University of Edinburgh
Edinburgh EH8 9AB, United Kingdom
iain.wallace@ed.ac.uk

Michael Rovatsos
School of Informatics
The University of Edinburgh
Edinburgh EH8 9AB, United Kingdom
michael.rovatsos@ed.ac.uk

ABSTRACT

Reasoning about others, as performed by agents in order to coordinate their behaviours with those of others, commonly involves forming and updating beliefs about hidden system properties such as other agents' mental states. In this paper we introduce the *Expectation-Strategy-Behaviour* (ESB) framework which provides a generic machinery for such practical social reasoning and can be easily coupled with deliberative, knowledge-based architectures such as BDI. We present a conceptual model of ESB, its formal semantics, and simple initial reasoning algorithms that illustrate how the principles of ESB can be used to implement bounded "social" rationality in multiagent designs. A case study is used to show how ESB substantially simplifies the design of agents that include social reasoning functionality.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms, Design, Theory

Keywords

Architectures, Methodology, Expectations, Social Reasoning, Agent-Oriented Software Engineering, BDI

1. INTRODUCTION

Social reasoning, i.e. reasoning about others in order to better coordinate one's behaviour with theirs, plays a crucial role in achieving intelligent decision making in multiagent systems. In the general case the internal state of agents is usually not observable to other agents, e.g. in systems in which agents pertain to different users unwilling to reveal all details of their agent's internal design. As such, social reasoning mostly concerns hidden properties of the system, i.e. beliefs about non-observable features of the system such as other agents' beliefs and goals, their trustworthiness, etc.

From an agent design point of view, specifying what an agent should think about others in specific circumstances requires the specification of reasoning rules that define how beliefs about these *non-observable* properties will be formed and updated based on *observable* features of the overall behaviour of the system (such as environmental state changes, agent actions, etc). So what is needed are belief revision [4] mechanisms that work in practice for the agent when interacting with others and in accordance with its local beliefs, its objectives, and its deliberation and planning mechanisms.

Such belief revision strategies would normally be integrated in traditional practical reasoning architectures such as BDI [2, 7] within the belief update functions of the architecture, which forms part of the sense-reason-act cycle of any architecture suitable for use in agent-based systems.

In this paper, we argue that singling out belief update rules for social reasoning from general belief revision mechanisms can be beneficial because

1. it enables us to apply *bounded rationality* principles to social reasoning (in a similar way as these are applied to general practical reasoning in architectures such as BDI), and

2. it simplifies the agent implementation process from the designer's point of view by providing automated support specific to reasoning patterns that are common in social reasoning.

To support these claims, we report on a framework for practical social reasoning called the *Expectation-Strategy-Behaviour* (ESB) architecture which we have developed and which provides the abstractions and reasoning mechanisms that are necessary to realise the suggested benefits. A particular driving factor behind the design was that the techniques developed can be used in a practical way to help agent system designers, and to build a coherent reasoning machinery.

ESB is fundamentally based on the notion of *expectations* as beliefs regarding hidden properties of the system that contain a specification of a "test" that will be used to verify whether the property held or not. They also contain "responses" which describe how the agent will update its beliefs depending on the outcome of the test (where different expectations in the agent's overall "expectation base" may affect each other). We argue that expectations, together with *strategies* (which describe the style of reasoning that the agent applies to its expectation base) and *behaviours* (that capture how the currently held expectations affect actual agent behaviour) provide us with a natural way for describing concrete social reasoning methods and for devising modular social reasoning designs.

Cite as: Bounded Practical Social Reasoning in the ESB Framework, Iain Wallace, Michael Rovatsos, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1097–1104

Furthermore, use of ESB enables us to develop relatively simple automated analysis algorithms that can be utilised by the human designer to support the design process. We will illustrate this by proposing initial graph-based algorithms that can be used in this way.

It should be emphasised that ESB does *not* propose a concrete social reasoning method. Such methods, based on (*inter alia*) concepts such as norms and social laws [3], commitments, trust and reputation [6, 5], or opponent modelling [8], abound in the literature [9, 11]. Instead, ESB is based on the very generic idea of "managing belief revision strategies about hidden properties in a boundedly rational but socially meaningful way" that could be applied to any beliefs concerning social reasoning concepts like those listed above. In this way, it aims at achieving a level of generality similar to that of "meta-architectures" like BDI, while at the same time capturing all the elements that are frequently used by social reasoners to support the process of designing them.

The remainder of this paper is structured as follows: In section 2, we introduce the ESB architecture informally. This is followed by the presentation of a formal model of the framework in section 3. Section 4 illustrates the workings of the ESB model with an example, which is used in section 5 where we introduce simple reasoning algorithms for our framework. Section 6 concludes, and section 7 gives an outlook to future work.

2. THE ESB ARCHITECTURE

The ESB architecture is an abstract model for practical social reasoning systems based on the concept of *expectations*, which we can define as follows:

> An *expectation* is a *conditional belief* regarding a statement whose truth status will be eventually *verified* by a *test* and *reacted* upon by the agent who holds it.

More concretely (and as will be defined formally below), we assume that an expectation concerns a belief Φ held by an agent A under condition C. Depending on the outcome of a test T, the agent will take action ρ^+ if the expected belief was confirmed (positive response), and ρ^- if not (negative response).

As an example, consider an agent A who expects another agent B to execute some action upon promising to do so. The expected belief is "B will perform an action (Φ) if he has promised to do so (C)", the test (T) consists of observing either the action itself or its consequences. The reaction might be, for example, that A will reward or punish B depending on whether the promise is kept (ρ^+) or not (ρ^-), respectively; or, the expectation *itself* might be modified after having observed that B performs the action or not, for example by increasing or decreasing the level of trust towards B.

Why should expectations be an important concept for social reasoning? We argue that this is because they capture all the elements that are normally present in systems that reason about maintaining and revising beliefs regarding parts of the system that cannot be observed directly. Intuitively, if Φ is a hidden property of the system (e.g. "agent B is trustworthy" in the example above), the reasoning agent A would make its belief in this statement contingent on some condition C under which it is relevant (simply to split a potentially large set of possible beliefs into manageable subsets each of which is only relevant under certain circumstances), and it would specify circumstances T under which the belief would be reaffirmed or negated. Finally, the agent's design would have to specify what to do if the test succeeds (ρ^+), and what to do if it fails (ρ^-).

Another property of such social reasoning rules is that different statements about expectations tend to be specified in a disconnected (or, to put it more positively, *modular*) way in practice: In the above example, one might specify "if B does not perform the promised action, I will start thinking that he is a liar" and "if B has kept his promise on more than 80% of all occasions, he is not a liar" separately, as these reflect "rules of thumb" regarding our assumptions about others' internal properties and the heuristics we apply to guess these properties from observed behaviour. This intuition leads us to believe that individual social reasoning rules can be self-contained and there may be in fact situations where a bunch of these rules are put together in a system with no regard for consistency between them.

Strategies, on the other hand, specify ways in which sets of expectations are processed. The easiest way to conceptualise this process is to think of all possible sets of expectations that could arise from future observations of test outcomes, and to think of a strategy as a particular way of traversing the graph that results from mapping out all possible future expectation-relevant events. This is where *bounded rationality* comes into play, as different strategies allow for different levels of complexity in projecting possible future beliefs. Simple strategies include bounding the depth of predictions, or excluding expectation sets that are unreachable from the initial state, or impossible because of mutually exclusive test outcome. More complex, and truly "social" ones include, for example, assuming that different states have different utilities and applying game-theoretic reasoning in the exploration of future expectation states. Note that concrete agent designs may of course allow for switching strategies at runtime.

The purpose of strategies is to determine what model of belief revision the agent will use to evaluate the conditions of *behaviours*. Generally speaking, behaviours are rules that determine how the state of one's overall expectation base affect the reasoning agent's behaviour. In practice, they will usually be conditioned on statements about expectations, and the truth value of these will be established applying the agent's strategy to the expectation base. One behaviour in the above example might be, for instance, "if A thinks that B is unreliable, A doesn't have to be trustworthy to B". This rule refers only to the current state, but more complex rule conditions like "if it is possible that B might be deemed unreliable in the future" might involve evaluating expectations along a set of possible projections of future events (and this is where the complexity of the strategy would strongly determine how elaborate the reasoning process is that the agent has to apply to verify behaviour conditions).

In itself, ESB does not of course generate any concrete agent behaviour and has to be integrated with some more general reasoning and execution architecture to yield a complete agent design. For the purposes of this paper it suffices to assume that behaviours always have an overall belief change in the agent as their only consequence, i.e. each ESB behaviour is of the form "**if** ψ **then** (don't) believe ϕ" where ψ is a condition on the expectation base. This means that a behaviour simply causes a belief update, and we assume

that an architecture such as BDI will be affected by this update appropriately, e.g. by making certain plans available or unavailable depending on the truth status of ϕ.

3. FORMAL MODEL

Formally, the expectation "whenever condition C holds, agent A will expect Φ, verify it with test T and react with ρ^+ if it is true or ρ^- otherwise" is represented as

$$\mathbf{Exp}(N\ A\ C\ \Phi\ T\ \rho^+\ \rho^-)$$

where

N is the *name* of the expectation, taken from a set of expectation labels $\{N, N', \ldots\}$,

A is the *agent* holding this expectation, taken from a set of agent names $\{A, A', \ldots\}$,

C is the *condition* under which the expectation holds,

Φ is the expected belief, i.e. an event or an expectation that another agent is assumed to hold[1], or some other inferred belief about the world,

T is the *test* which confirms (or rejects) the expectation of Φ (an observable event),

ρ^+/ρ^- are the positive/negative *responses* to the test T; these could be modifications to its internal state, including modifying the set of expectations it holds.

The formal semantics of expectations can be easily expressed in the $\mathcal{LORA}$ [10] logic, a multi-modal propositional BDI logic which combines modalities for belief, desire and intention with aspects of dynamic logic and CTL temporal logic, using standard propositional notation. Using $\mathcal{LORA}$, we can define expectation tuples as follows:

$$\mathbf{Exp}(N\ A\ C\ \Phi\ T\ \rho^+\ \rho^-) :=\mathtt{A}\Box(\mathit{current}\ \mathcal{U}\ \mathit{update})$$

where

$$\begin{aligned}\mathit{current} =&(\mathbf{Bel}\ A\ C) \Rightarrow (\mathbf{Bel}\ A\ \Phi)\\ \mathit{update} =&\Big((\mathbf{Bel}\ A\ T) \wedge \mathtt{A}\bigcirc \rho^+\Big)\vee\\ &\Big((\mathbf{Bel}\ A\ \neg T) \wedge \mathtt{A}\bigcirc \rho^-\Big)\end{aligned}$$

and

$$\begin{aligned}\rho^+ \equiv&\{\mathbf{Exp}(N'\ A'\ C'\ \Phi'\ T'\ \rho'^+\ \rho'^-), \ldots\}\\ \rho^- \equiv&\{\mathbf{Exp}(N''\ A''\ C''\ \Phi''\ T''\ \rho''^+\ \rho''^-), \ldots\}\end{aligned}$$

such that the responses are sets of expectations.

To avoid going into too many details of $\mathcal{LORA}$ here, it suffices to point out that $\mathtt{A}$ is a path quantifier that refers to the temporal structure of the logic and denotes "on all paths", $\mathcal{U}$ stands for the "until"-modality (so that $\phi\ \mathcal{U}\ \psi$ means ϕ will hold in the future until ψ, and ψ will occur at some point), **Bel** is the "believes"-modality, $\bigcirc$ means "in the next time step", and $\Box$ denotes "always in the future". Φ and C are arbitrary $\mathcal{LORA}$ formulae, T is an observable event that becomes true or false at some point in the future and is beyond the agent's control.

Thus, $\mathbf{Exp}(N\ A\ C\ \Phi\ T\ \rho^+\ \rho^-)$ informally means:

> *In all paths it is always the case that: Belief in C implies belief in Φ, until T is believed and in all paths at the next time step ρ^+ or $\neg T$ is believed and ρ^- holds.*

Below, we will assume that "A holds expectation N" has precisely this meaning.

To refer to the various parts of an expectation and the various different expectations an agent may hold, we introduce the notation E^i_X to refer to component X of expectation i in a set of expectations. So, for example, E^2_C refers to the condition of expectation 2.

Furthermore, we define **EXP** as the total set of all possible expectations a particular agent might have, which is currently taken to be finite[2]. Note that by this we mean all expectations that are possible for *one particular agent* to hold, not the set of all possible expectations that can be expressed using the syntax introduced.

The following two subsets of **EXP** will also be useful:

- At any time an agent holds a certain number of so-called *active* expectations i.e. exactly those expectations for which $(\mathbf{Bel}\ A\ C) \Rightarrow (\mathbf{Bel}\ A\ \Phi)$. These form the set $EXP_A \subseteq \mathbf{EXP}$.
- An expectation depends on a condition C, under which it holds. The set of expectations whose conditions are currently true, i.e. those for which $(\mathbf{Bel}\ A\ C) \Rightarrow ((\mathbf{Bel}\ A\ \Phi) \wedge (\mathbf{Bel}\ A\ C))$ holds, form the *current* expectation set $EXP_C \subseteq EXP_A$.

The set of active expectations changes according to the results of tests and responses as defined in the expectations themselves. Responses need only add or remove expectations (as this is equivalent to making changes in expectations, if the total set is finite), and below we will use $add(S)$ amd $remove(S)$ as the only two possible responses in an expectations which correspond to removing the set S from the set of currently held expectations.

3.1 Strategies

Next, in order to define strategies formally, consider the set of active expectations $EXP_A \in \wp(\mathbf{EXP})$ which is changed by the responses, whenever test outcomes become available. So the responses can be thought of as defining a relation between the states that the agent's reasoning can be in:

$$xRy \text{ where } x, y \in \wp(\mathbf{EXP})$$

The agent can then be in one of a finite number of expectation states which correspond to possible active expectation sets. These states and the relation between them can be represented intuitively by a digraph referred to as the *expectation graph*. More useful still is a restricted sub-graph, containing only those vertices accessible from the initial state (we will see an example of this later in Figure 1).

With this structure in mind, a strategy is a way to restrict this graph in some way, according to some reasoning scheme. This is useful as behaviours rely on checking conditions on the graph. For example, a strategy could be only considering the positive responses of expectations, which would restrict the accessible portions of graph. The abbreviation S_i will be used below to refer to a single state i where there are several states under discussion.

[1]Nesting expectations allows modelling of other agent's mental states, and adaptive behaviour on the part of either agent.

[2]This restriction has certain implications; for example, responses that increment a counter cannot be included.

3.2 Behaviours

As we have explained above, we assume that behaviours are rules matching conditions on the expectations to actions external to the social reasoner, where actions would take the form of adding or removing beliefs to influence (say) a BDI practical reasoner (by restricting the plans in the library that can be currently used).

The set of expectations describes not only current social reasoning – represented in the active expectations – but also possible future belief states, and these belief dynamics are encoded in the graph structure. So behaviours may include conditions such as "Φ will always be expected" and "it is possible to expect Φ" in the future.

For now, we are just considering expectations at the meta-level, rather than reasoning over their content. For example, consider a simple case where $\Phi^1 = p$, $\Phi^2 = q$ and $p \Rightarrow q$. Should behaviours which have Φ^2 as their condition also apply when the agent holds Φ^1? For the time being, we argue that expectations should be considered as atoms, and inferences between them not drawn. This reduces the power of the system (for example it may not be possible to check for certain expectation conflicts) but it also greatly simplifies the system. And it is in part due to increased simplicity of reasoning that we expect to see some advantage in using ESB, where in fact it does not require the use of logic in actual designs (although it would be fairly straightforward to come up with formal inference rules that are based on tautologies in $\mathcal{LORA}$ and allow reasoning over relational expectation contents, albeit with the usual tractability problems associated with such deductive reasoning in practice).

4. INTRODUCING AN EXAMPLE

To illustrate the workings of ESB we use an example scenario taken from the domain of the Rummy [1] card game. For the purposes of this example it is only important to understand that players are trying to collect either runs of cards in a suit, e.g. {2♣,3♣,4♣}, or sets, e.g. {2♣,2♢,2♡} (given a card C, we refer to other cards that could be part of a set/run with that card as a "member of C's sets/runs"). *Melds*, as these sets or runs are called, must contain a minimum of three cards. On their turn a player must either pickup a card from the deck, which is unseen, or from the top of the discard pile, which is visible to all. They then create any melds they can and place a card on the top of the discard pile, play then passes to the next player.

In our example expectations shown in Table 1, the responses take the form of adding and/or removing the specified expectations as indicated, depending on the T test outcome. Combining the sets of expectations to remove and add (assuming they are disjoint) is enough to specify a state transition. The tests are listed with the positive and negative result, so when we talk about T holding, we mean "if + and not −". The initial set of expectations held are $\{E^1, E^2, E^5\}$. This represents an agent whose starting position is to be open to the notion of the opponent carrying out either strategy. As the purpose of this example is to illustrate the workings of ESB, the rules have been restricted to specific examples for simplicity. For example, it should be obvious that 2 and 5 are both instances of some more general rule – to expect the opponent is collecting a run when they pickup a member card of the run.

The card game Rummy provides a simple example domain, as the strategies that an opponent might use are known, but defined by the cards they hold – which are unknown. Therefore assumptions (Φ) must be made about these cards based on what the opponent has picked up and/or discarded in the past (C). These assumptions are borne out by the opponent's future plays (T) and the reasoning can be adjusted (ρ^+, ρ^-). Behaviour can then be chosen to be based not only on current expectations, but also on potential future melds they may collect – accounting for an opponents possible change in reasoning in the future.

4.1 ESB Setup

The example involves only two players, where the ESB agent A reasons about its opponent B picking up a 2♢. We consider only relatively naive reasoning on A's part: that B will either be collecting a run of diamonds around 2, or a set of 2s (A assumes B won't hedge her bets by collecting cards for both a run and a set). The reasoning for this example is encoded in the expectations in Table 1.[3]

We assume that the game is expected to end soon in the current situation. For this reason, A has decided not to consider all possible future strategies, but to only reason about what B's strategy might be at this time.

A simple strategy we can apply in this case is to only consider the expectation graph to a depth of one from the current state and to ignore loops. The justification of this strategy is that responses are triggered by observations (which update the status of tests), and these must come from observing cards being played or picked up. For the purposes of checking if behaviours are possible in the future loops are not considered, as those expectations are already in the current set of expectations and accounted for.

Behaviours take the form of conditions (on the strategy graph and expectations) and actions (which may be direct actions or internal ones). For the purposes of this example, the agent cares only which cards are safe to discard based on the model of the other player's reasoning. So three simple behaviours are as follows:

1. Only discard a 2 if B is not collecting 2s.
2. Only discard a 3 if B is not collecting 3s.
3. Only discard cards I don't expect B to be collecting in the future.

Whilst these make sense intuitively, it is worth translating them into specific conditions on the expectations, as this will prove useful for the rest of the example:

1. (a) Discard 2♢ if $\neg E^1_\Phi \wedge \neg E^2_\Phi \wedge \neg E^5_\Phi$
   (b) Discard 2♣ if $\neg E^1_\Phi$
2. Discard 3♢ if $\neg E^2_\Phi \wedge \neg E^5_\Phi$
3. (a) Discard 2♢ if in all accessible states $\neg E^1_\Phi \wedge \neg E^2_\Phi \wedge \neg E^5_\Phi$
   (b) Discard 2♣ if in all accessible states $\neg E^1_\Phi$
   (c) Discard 3♢ if in all accessible states $\neg E^2_\Phi \wedge \neg E^5_\Phi$

[3]As there are limited actions available to the players, and both tests and conditions must depend on these actions, the T and C components appear similar in the Rummy example, this need not be true in the general case.

Table 1: Expectations in the Rummy example: the labels of initial expectations (1,2,5) are shown in bold face; tests are broken down into $+(\ldots)$ and $-(\ldots)$ events that would confirm/disappoint the expectation

N	C	Φ	T	ρ^+	ρ^-
1	B picked up 2♢	B is collecting 2s	$+(B$ picks up a 2) $-(B$ ignores/discards a 2)	remove({2,5}) add({3,4})	remove({1})
2	B picked up 2♢	B is collecting ♢ run	$+(B$ picks up member of 2♢ run) $-(B$ ignores/discards member of 2♢ run)	remove({1,3,4}) add({5})	remove({2,5}) add({3,4})
3	B discarded 2♣	B not collecting 2s	$+(B$ ignores a 2) $-(B$ picks up a 2)	-	remove({1,3}) add({2,5})
4	B ignored a 2	B not after 2s for run or set	$+(B$ ignores a 2) $-(B$ picks up a 2)	-	remove({1,3}) add({2,5})
5	B picked up 3♢	B is collecting ♢ run	$+(B$ picks up another member of 3♢ run) $-(B$ ignores/discards member of 3♢ run)	remove({1,3,4}) add({2})	remove({2,5}) add({1,3,4})

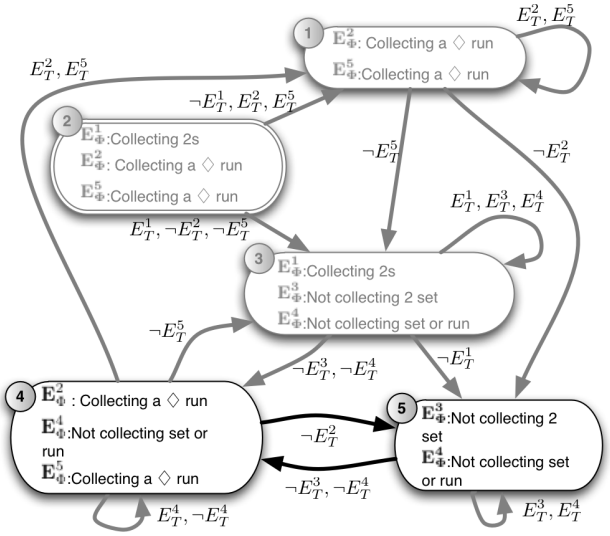


Figure 1: Accessible Expectation Graph

Here, "in all accessible states" refers to all states accessible from the current expectation state (assuming all E_Φ are true), according to the strategy. So in the case of this example, it means all vertices adjacent to the one corresponding to the current state.

As far as the link to a practical reasoning is concerned, in the case of this example, the behaviour "action" "Discard 2♢" could represent a belief that it was safe to discard the 2♢. Guards on plans of, say, a BDI reasoner, would then ensure that only actions consistent with the permitted behaviours (the outcome of ESB reasoning) were selected.

4.2 The Accessible Expectation Graph

Figure 1 shows the accessible expectation graph generated from the expectations in Table 1. Vertices are labelled with the expected belief of each expectation tuple, as this makes consideration of the relevant behaviours easier. The initial state S_2 is indicated by the double lines. Edges represent transitions specified by the responses. A loopback to a state indicates that the $\rho^{+/-}$ responses of some expectations do not cause a change in state. The bold parts of the graph illustrate the restriction to the strategy graph for an agent in state S_5.

5. REASONING IN ESB

To show that ESB can take account of typically complex social reasoning problems alongside practical reasoning, we will now describe how the agent's social reasoning mechanism can be bounded using ESB, and how some aspects of reasoning within ESB can be automated in a generic, domain-independent fashion.

5.1 Simple Reasoning Algorithms for ESB

The central structure in ESB is the expectation graph, or its derived variants. However it is not necessary for a system designer to create this, a lot of the power of ESB comes from the fact that it is a general structure and can be generated simply from a list of expectations. To create the complete expectation graph – featuring the complete set of states is a fairly simple (even though computationally expensive) process. The basic idea is that each possible combination of expectations forms a vertex, and where a response is to add some set of expectations and remove some set, an edge links that vertex to identical one only with the added expectations, and without the removed set. This can be done with the following algorithm:

```
Initialise set of vertices:  V ← ℘(EXP)
FOR each state S_a in V
    FOR each expectation E^X in S_a
        FOR each response ρ^+, ρ^-
            Response is add(add_set), remove(rem_set)
            Add a directed edge from S_a to S_b,
            Where add_set = S_b \ S_a, rem_set = S_a \ S_b
            Label the edge with E^X_T for ρ^+
            Label the edge with ¬E^X_T for ρ^-
        END FOR
    END FOR
END FOR
```

This algorithm represents the relation between states described previously in section 3.1. As this is a naive algorithm that creates the complete expectation graph (with $2^{|\mathbf{EXP}|}$ states) it will be impractical for most real-world situations. A very basic simplification one can perform to limit its size is to consider the *accessible* expectation graph, i.e. the graph generated by a search that starts in the initial state and expands the graph until all the edges have been added in the manner described above. Termination occurs when every expectation in every state has been considered, and no more vertices are added. Figure 1 shows this accessible graph for our Rummy example.

As stated above, we assume that in the situation in question, the agent simply restricts the graph to a depth of one, i.e. the current state and all its adjacent states. This can be easily obtained from the expectation graph described above, given a current expectation state (for state S_5 in our example this sub-graph is displayed as the darker areas of the expectation graph in Figure 1). A key point worth noting is that while the expectation graph is fixed, the strategy graph is dynamic and may change based on the current expectation state (and, potentially, a change in strategy).

5.1.1 Implementation as an FSM

Independent of the actual expectations themselves, a large part of the ESB design can be generalised for implementation. To handle tracking state, a finite-state machine can be constructed with one state for each state S in the accessible expectation graph, and selecting transitions by applying the E_T tests for any current expectations (exactly those with $E \in S$ and $E_C = \text{true}$). As only the expectations current to the state need be checked (since all their dependencies have been "precompiled" into the expectation graph offline), this bounds run-time reasoning further in useful way.

As well as keeping track of the ESB state, it is necessary to keep track of the expectations to test. This is another area where the ESB approach helps bound agent reasoning, as it is only necessary to test the conditions of the expectations in the current state, and infer the appropriate expected beliefs Φ, as follows:

```
FOR all E^X in current state S_i DO
    IF E^X_C THEN Believe E^X_Φ
END FOR
```

The final part of an ESB design concerns the behaviours, and these may depend on the current state, or may be conditional on possible future expectations. For the current state it is simple to use the various Φ as the guard on plans at the level of the general practical reasoning – as in the example behaviours given above – but the future expectations require a slightly more complex approach. These conditions are likely to take the form of "in all possible cases the agents might expect X" or "in some possible case the agent might expect X". For the latter, it is simply a case of searching the strategy graph representation for a state matching the condition. For the "all cases" condition, the graph must be searched until all vertices have been checked.

5.2 Simplifying Behaviours

As a further example of how our graph-based representation of the reasoning structure provides opportunity for an analysis of the design – particularly to bound reasoning further – we introduce a method that can be used to reduce the number of behaviour rules that need to be checked each reasoning cycle, and the complexity of checking them.

More specifically, in our Rummy example the rules under 3. are expensive to check, as they involve searching the expectation graph – but these are typical of any behaviour that is conditioned on potential future events. The approach we suggest is to perform offline reasoning at the design stage, and will lead to simplifying run-time reasoning during execution. This off-line reasoning is of course itself computationally expensive, but we argue that performing this computation once during design is preferable, as it greatly simplifies reasoning during execution.

```
FOR each behaviour B create a new set
    of conditions B'_C containing only false
FOR each behaviour B
    FOR each state S_i = {E^1...E^n} ∈ accessible E-graph
        IF B_C= true for each combination of
           {E^1_Φ...E^n_Φ} THEN
              Add ∨S_i to B'_C
        ELSE IF B_C = false for each combination of
           {E^1_Φ...E^n_Φ} THEN
              Add nothing
        ELSE some minimal combination Min is a
           minimal subset of ({E^1_Φ...E^n_Φ} ∪ {¬E^1_Φ...¬E^n_Φ})
           making B_C = true THEN
              Add ∨(S_i ∧ Min) to B'_C
        END IF
    END FOR
END FOR
```

Figure 2: Algorithm to update behaviour conditions

Behaviours are conditions with actions linked to them, so optimisation takes the form of re-writing the conditions to include state, and where possible restricting the conditions yet further to simplify testing. The intuitive reasoning behind the algorithm we use is that considering each behaviour in each state in turn, there are three possibilities:

1. The behaviour condition is true regardless of the combination of expected beliefs that holds. In this case we can always apply the behaviour in this state.
2. The behaviour condition is false regardless of the combination of expected beliefs that holds. In this case we can never apply the behaviour in this state.
3. Whether or not the behaviour condition applies depends on whether some minimum combination of expected beliefs holds.

The main reason a behaviour condition might always be false (or true) is that it depends on future expectations – which are assumed to be true for the purposes of checking conditions – so only accessibility will matter. In this case it is the state that is important, so including it in the condition can simplify testing. The prospect of dependence solely on state is likely to be common, as it can affect any behaviour rules acting over change in reasoning.

The process of re-writing conditions to include state can be carried out with the algorithm shown in Figure 2. It is possible to have nothing added in the second case, as for every other case the state is considered, so states left out of the condition term will satisfy no other part of the term, and will not be considered at all. Applying this algorithm to the current Rummy example, we obtain the simplified set of behaviours shown in Table 2.

The process does not end here. As some of the behaviours share identical actions, they can be combined by only allowing the action when both conditions hold. This brings us closer to the original meaning of behaviour rule 3, which was previously expanded to form the three separate sub-rules. This also greatly simplifies the conditions, as for example, forming a conjunction between the conditions of 1(a) and 3(a) includes the term $\wedge false$, thus reducing the entire term to $false$. We can easily build another table that lists each new behaviour against every state as well as the condition

Table 2: New Behaviour Conditions after Expectation Graph Analysis

Behaviour	Action	Original B_C	New B'_C
1(a)	Discard 2♢	$\neg E^1_\Phi \wedge \neg E^2_\Phi \wedge \neg E^5_\Phi$	$(B_C \wedge S_2) \vee (\neg E^2_\Phi \wedge \neg E^5_\Phi \wedge S_1) \vee (\neg E^1_\Phi \wedge S_3) \vee (\neg E^2_\Phi \wedge \neg E^5_\Phi \wedge S_4) \vee S_5$
1(b)	Discard 2♣	$\neg E^1_\Phi$	$S_1 \vee (\neg E^1_\Phi \wedge S_2) \vee (\neg E^1_\Phi \wedge S_3 \vee S_4) \vee S_5$
2	Discard 3♢	$\neg E^2_\Phi \wedge \neg E^5_\Phi$	$(\neg E^2_\Phi \wedge \neg E^5_\Phi \wedge S_1) \vee (\neg E^2_\Phi \wedge \neg E^5_\Phi \wedge S_2) \vee S_3 \vee (\neg E^2_\Phi \wedge \neg E^5_\Phi \wedge S_4) \vee S_5$
3(a)	Discard 2♢	In all accessible states $\neg E^1_\Phi \wedge \neg E^2_\Phi \wedge \neg E^5_\Phi$	$false$
3(b)	Discard 2♣	In all accessible states $\neg E^1_\Phi$	$S_3 \vee S_5$
3(c)	Discard 3♢	In all accessible states $\neg E^2_\Phi \wedge \neg E^5_\Phi$	S_1

required to trigger it. Each entry is either false, or where there is a conjunction of a state and condition, the condition is added to that state and behaviour. Where only a state is present, the condition can be considered to be *true*. For the previous example, this produces the following table (cells indicate the condition B'_C):

Action	Discard 2♢	Discard 2♣	Discard 3♢
State			
1	*false*	*false*	$\neg\mathbf{E^2_\Phi} \wedge \neg\mathbf{E^5_\Phi}$
2	*false*	*false*	*false*
3	*false*	$\neg\mathbf{E^1_\Phi}$	*false*
4	*false*	*false*	*false*
5	*false*	**true**	*false*

This allows us to infer that we only need to check behaviour conditions in states 1 and 3, and that state 5 always triggers behaviour 1(b). The final simplified behaviours are visualised in a simplified graph shown in Figure 3.

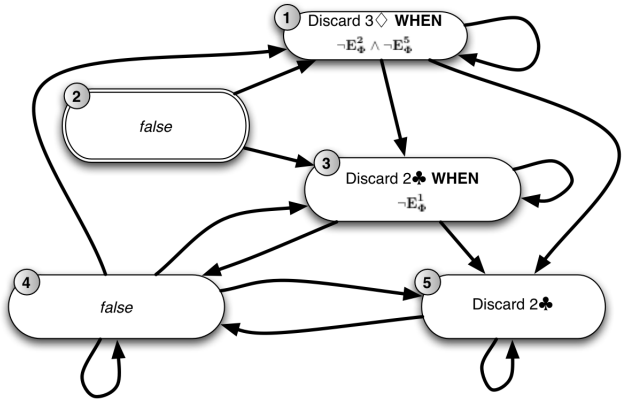


Figure 3: Reduced Behaviour Expectation Graph

5.3 Advanced Social Strategies

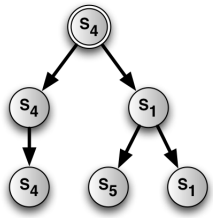
The strategy in the example presented above, which is to simply limit graph search by depth, is very simplistic and does not use truly "social" approaches to bounded practical reasoning about the other.

ESB can be used to represent much more complex strategies that allow us to incorporate considerations about distributed, multi-party reasoning. When nested expectations are allowed, for example, strategies could restrict the depth of nesting, or perhaps even restrict nesting based on some measure of confidence (which could be updated by responses).

Another possibility which we only briefly discuss here for lack of space is using ideas from decision theory or game theory to influence the design of strategies. For a short example, consider an agent wishing to adopt a minimax strategy for reasoning. The key point of such a strategy is that at every move it is assumed the opponent will pick the worst scoring option for the agent, and so for its move the agent will maximise this minimum. To apply this concept to our running example, the "moves" are the observed results of the T tests. The ESB agent's moves are not represented in the expectation graph for simplicity, but its behaviour should maximise this worst-case play by the opponents. We restrict the expectation graph by applying this principle before evaluating the preconditions of behaviours. This requires making assumptions about the evaluation function the opponent uses, which (to keep things simple) might look something like this:

- A pickup scores -1, as they've gained a useful card.
- A discard scores $+1$ as they've lost a card.
- Ignoring a card scores 0 as it tells us nothing.

So the strategy is now to build a tree of the graph to a certain maximum depth (say 2), and then restrict it to only those paths which score lowest (adding the scores down each path to gain a total at the leaves). Assuming state S_4 as initial state, this gives us the following strategy graph:



For lack of space, this strategy does not include assumptions about the agent's own behaviour. This means that the only influence the mini-max strategy has is to restrict behaviours according to it. If the agent's actions were included in the example, then the strategy graph could be reduced further still by selecting those which maximise the ESB agent's own utility. It does however give a flavour of what more advanced social reasoning strategies might look like, and illustrates the power of the framework keeping in mind that we could apply the algorithms described above to such more advanced mutual modelling schemes.

6. CONCLUSIONS

In this paper we have introduced the ESB framework for the design and implementation of a practical social reasoning component for an intelligent agent. Based on the idea of constructing social reasoning mechanisms from belief revision mechanisms regarding hidden properties in the system (such as the mental states of other agents), we developed a notion of *expectation* that captures all the elements that are necessary to link beliefs held about non-observable parts of the system. It also allows us to express belief dynamics at a practical, procedural level while benefiting from declarative representations. The other two main components of the ESB architecture are (i) *strategies* which determine how expectations are reasoned about and to what extent the agent is bounding its own social reasoning and (ii) *behaviours* which link expectation states to the more general practical reasoning component of the agent.

Using a simple graph-based approach, we then went on to describe generic reasoning methods which prove ESB allows generalisation of part of a social agent's design, and aid in the analysis and bounding of its reasoning.

Encoding the reasoning in expectations appears intuitive, and given this encoding the algorithms described above allow for the expectation graph, strategy graph and resulting state-update FSM to be generated, regardless of the content of the expectations themselves. It is easy to see how this could be extended to allow direct execution of an ESB specification, through automated generation of the FSM.

More designer input is required in the case of the behaviours, which must be translated into conditions on the expectations, and future expectations. However we argue that this is a logical way to design an agent, as much like determining plans in a BDI agent design, ESB behaviours follow from reasoning of the form "in what conditions do I perform behaviour X?".

7. FUTURE WORK

The most obvious avenue to pursue for future work is the development of more algorithms for analysis of the agent design, either to compare properties of reasoning schemes, bounding an agent's reasoning, or identifying possible defects in the design – in a similar fashion to a model checker. As it stands, we do not exploit logical inference between expectations which limits the power of expectations. If ESB was extended to exploit inference, then complex logical behaviour conditions could be evaluated using a model checker and language such as MABLE [12].

Model checking is also a possible solution to a problem with the work – there is a slightly inconsistent mix of the graph-based approach and the logic-based definition of what it means to hold an expectation. A possible solution for this is to prove that algorithms applied to the graph are consistent with respect to the logic.

It is also planned to extend ESB to include nesting of expectations. The theory has been designed with this in mind, as it shows promise to be a very powerful concept. This would involve the expected belief Φ of an expectation being not just a simple belief, but an expectation itself, possibly of another agent. This would in turn allow for more explicit modelling of other agent's reasoning processes. The notion of reasoning with expectations belonging to another agent is easy to see, even for the given example. Consider that the expectations could be reversed, so that they were owned by the opponent, and then the agent's behaviours based on these could be chosen to ensure that it played appropriately to ensure the other agent was in a certain expectation state. This provides a mechanism to deliberately influence the other.

The possible explosion in the number of states in the expectation graph is another issue that requires further investigation. Restricting to the accessible expectation graph in the current state will not always help, as this is still the full graph in the worst case. The computational cost of the algorithms presented here is high, however these are only the most obvious naive algorithms – so improvements are certainly possible.

As ESB is only targeted at the social reasoning component of an agent, and a large part of this can be generalised and automated, it would seem that ESB could at least offer a more suitable solution than an all-in-one reasoner. This does not solve the complexity problem however, and for any implementation bounding the graph for computational reasons will be important. This is where strategies aimed at reducing the scope of the graph in use could come into play.

These possibilities all fall within one general avenue of research – that of ESB as a tool to aid design. The other key avenue to explore is that of ESB as a tool to aid implmentation, where ESB specifications are directly executed in some way. This would be possible, as a large part of the process is both general and easily automated as described here. These (and more refined) algorithms describe a kind of "ESB engine" which could be further developed into an implementation as an extension to an existing practical agent reasoning scheme.

8. REFERENCES

[1] Rummy.com - The Rules of Rummy, 2008. http://rummy.com/rummyrules.html.

[2] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[3] F. Dignum, D. Kinny, and L. Sonenberg. From Desires, Obligations and Norms to Goals. *Cognitive Science Quarterly*, 2(3-4):407–430, 2002.

[4] P. Gärdenfors, editor. *Belief Revsion (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, 1992.

[5] B. Horling and V. Lesser. A survey of multiagent organizational paradigms. *Knowledge Engineering Review*, 19(4):281–316, 2004.

[6] N. Jennings. Commitments and conventions: The foundation of coordination in multiagent systems. *Knowledge Engineering Review*, 8(3):223 – 50, 1993.

[7] A. S. Rao and M. Georgeff. An abstract architecture for rational agents. In W. S. C. Rich and B. Nebel, editors, *Proceedings of Knowledge Represenation and Reasoning (KR&R-92)*, 1992.

[8] S. Saha, A. Biswas, and S. Sen. Modeling opponent decision in repeated one-shot negotiations. In *Proceedings of the International Conference on Autonomous Agents*, pages 535 – 541, 2005.

[9] G. Weiß, editor. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA, 1999.

[10] M. Wooldridge. *Reasoning about Rational Agents*. The MIT Press, 2000.

[11] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.

[12] M. Wooldridge, M.-P. Huget, M. Fisher, and S. Parsons. Model checking for multiagent systems: The MABLE language and its applications. *Int. J. Artif. Intell. T.*, 15(2):195–225, 2006.