Semantic Email: Theory and Applications

Luke McDowell *, Oren Etzioni, and Alon Halevy

Department of Computer Science & Engineering, University of Washington, Box 325350, Seattle, WA 98195-2350, USA

Abstract

This paper investigates how the vision of the Semantic Web can be carried over to the realm of email. We introduce a general notion of semantic email, in which an email message consists of a structured query or update coupled with corresponding explanatory text. Semantic email opens the door to a wide range of automated, email-mediated applications with formally guaranteed properties. In particular, this paper introduces a broad class of *semantic email processes*. For example, consider the process of sending an email to a program committee, asking who will attend the PC dinner, automatically collecting the responses, and tallying them up. We define both logical and decision-theoretic models where an email process is modeled as a set of updates to a data set on which we specify goals via certain constraints or utilities. We then describe a set of inference problems that arise while trying to satisfy these goals and analyze their computational tractability. In particular, we show that for the logical model it is possible to automatically infer which email responses are acceptable w.r.t. a set of constraints in polynomial time, and for the decision-theoretic model it is possible to compute the optimal message-handling policy in polynomial time. In addition, we show how to automatically generate explanations for a process's actions, and identify cases where such explanations can be generated in polynomial time. Finally, we discuss our publicly available implementation of semantic email and outline research challenges in this realm. [1]

Key words: Semantic Web, satisfiability, decision-theoretic, reasoning with aggregation

[1] See http://www.cs.washington.edu/research/semweb/email for a publicly accessible server (no installation required); source code is also available from the authors.

* Corresponding author. Tel: +1-206-543-1695 Fax: +1-206-543-2969
Email addresses: lucasm@cs.washington.edu (Luke McDowell), etzioni@cs.washington.edu (Oren Etzioni), alon@cs.washington.edu (Alon Halevy).

1 Introduction

The Semantic Web envisions a portion of the World-Wide Web (WWW) in which the underlying data is machine understandable and can thus be exploited for improved querying, aggregation, and interaction [4]. However, despite the great potential of this vision and numerous efforts, the growth of the Semantic Web has been stymied by the lack of incentive to create content, and the high cost of doing so. Content owners are not motivated to create the structured representations necessary to contribute to the Semantic Web without seeing the immediate benefit of their significant efforts. In fact, this problem is not unique to the Semantic Web — the Database and Knowledge Base communities have long recognized that users shy away from their tools because the perceived benefits are outweighed by the conceptual difficulty and overhead of structuring data. Instead, users often resort to spreadsheets, structured files or just plain text. Ironically, in many cases the resultant lack of data management facilities, and reasoning capabilities, ultimately leads to more work for users down the road.

This paper explores this problem by identifying a pain point where the cost/benefit equation associated with structuring data can be changed dramatically. While the WWW is a rich information space in which we spend significant amounts of time, many of us spend as much or more time on email. With the exception of the generic header fields associated with each message, email messages typically do not have semantic features. While the majority of email will remain this way, this paper argues that adding semantic features to email offers opportunities for improved productivity while performing some very common tasks. We establish the theoretical foundations for *Semantic Email* and address some of the practical challenges associated with it via a completely implemented system.

To illustrate the promise of Semantic Email, consider several examples:

- **Information Dissemination:** In the simplest case, suppose you send a talk announcement via email. With suitable semantics attached to the email, sending the announcement can also result in automatically (1) posting the announcement to a web calendar, and (2) sending reminders a day before the talk.
- **Event Planning:** Imagine you are organizing a program committee meeting and you want to know which PC members will stay for dinner after the meeting. Currently, you must send out the question and compile the replies *manually*, leafing through emails one by one. With semantic email, the PC members can provide the reply in a way that can be automatically interpreted and compiled, enabling such planning to scale to much larger numbers of people. In addition, after a few days, unresponsive PC members can be automatically reminded to respond, and those who have said they're not coming to the PC meeting need not be bothered with this query at all. Alternatively, suppose that you are organizing a *balanced potluck*, where people should bring either an

appetizer, entree, or dessert, and you want to ensure that the meal is balanced. Here semantic email can help ensure that the potluck is indeed balanced by examining the replies and requesting changes where necessary.

- **Report Generation:** Suppose you need to collect projected budget numbers from a large set of managers. With semantic email, you could send a single email request and have the system automatically tabulate the responses, possibly requiring the values to satisfy certain individual or aggregate constraints. The system could then easily generate a spreadsheet report or integrate this data with other sources (e.g., prior budgets).
- **Auction/Giveaway:** Imagine you want to give away concert tickets that you cannot use. You would like to send out an announcement and have the semantic email system give out (or auction) the tickets to the first respondents. When the tickets are gone, the system should respond politely to later requests.

These examples are of course illustrative rather than exhaustive. In general, there are at least three ways in which semantics can be used to streamline aspects of our email habitat:

(1) **Update:** We can use an email message to add data to some source (e.g., a web page, as in our first example).
(2) **Query:** Email messages can be used to *query* other users for information. Semantics associated with such queries can then be used to automatically answer common questions (e.g., seeking my phone number or directions to my office).
(3) **Process:** We can use semantic email to manage simple but tedious processes that we currently handle manually.

Because email is not set up to handle these tasks effectively, accomplishing them by hand can be tedious, time-consuming, and error-prone. The techniques needed to support the first two uses of semantic email depend on whether the message is written in text by the user or formally generated by a program on the sender's end. In the user-generated case, we would need sophisticated methods for extracting the precise update or query from the text (e.g., [15,31]). In both cases, we require some methods to ensure that the sender and receiver share terminologies in a consistent fashion.

This paper focuses on the third use of semantic email to streamline processes, as we believe it has the greatest promise for increasing productivity and is where users currently feel the most pain. These processes support the common case of *asking* people a set of questions, *collecting* their responses, and *ensuring* that the results satisfy some set of goals. Some hardcoded email processes, such as the meeting request feature in Outlook, invitation management via *Evite*, and contact management via *GoodContacts*, have made it into popular use. Each of these commercial applications is limited in its scope, but validates our claim about user pain. Our goal in this paper is to sketch a *general* infrastructure for semantic email processes, and

to analyze the inference problems it needs to solve to manage processes effectively and guarantee their outcome.

Collaboration systems such as Lotus Notes/Domino and Zaplets offer scripting capabilities and some graphical tools that could be used to implement sophisticated email processes. However, these systems (as with typical workflow systems [44]) lack support for reasoning about data collected from a number of participants (e.g., as required to balance a potluck or ensure that a collected budget satisfies aggregate constraints). In addition, such processes are constructed from arbitrary pieces of code, and thus lack the formal properties that our declarative model provides. We describe these properties and the limitations of existing systems in more detail in Sections 3 and 5. Finally, messages in such systems lack the structured content (i.e., in RDF [32]) of semantic email, precluding automated processing by the recipient (e.g., to decline invitations for unavailable times).

Our work is the first to articulate and implement a general model of *semantic email processes* (SEPs). Our technical contributions are the following. Section 2 introduces a formalization for semantic email processes. The formalization specifies the meaning of semantic email processes and exposes several fundamental reasoning problems that can be used by the semantic email *manager* to facilitate SEP creation and execution. In particular, a key challenge is to decide when and how the manager should direct the process toward an outcome that meets the originator's goals. We address this challenge with two different formal models. First, Section 3 addresses this challenge by describing a model of *logical* SEPs (L-SEPs) and demonstrating that it is possible to automatically infer which email responses are acceptable with respect to a set of ultimately desired constraints in polynomial time. For this model, Section 4 also describes how to automatically generate explanations for the manager's interventions, and identifies cases where such explanations can be computed in polynomial time. Second, Section 5 describes a model of *decision-theoretic* SEPs (D-SEPs) that alleviates several shortcomings of the logical model, and presents results for the complexity of computing optimal policies for D-SEPs. Finally, Section 6 discusses implementation issues that arise for semantic email and how we have addressed these in our system, and Section 7 contrasts our approach with related work. The appendices provides proofs for all of the theorems given in the body of this paper.

2 Semantic Email Processes

Our formalization of SEPs serves several goals. First, the formalization captures the exact meaning of semantic email and the processes that it defines. Second, it clarifies the limitations of SEPs, thereby providing the basis for the study of variations with different expressive powers. Finally, given the formalization, we can pose several reasoning problems that can help guide the creation of semantic email processes as well as manage their life cycle. We emphasize that the *users* of SEPs are not expected to understand a formalization or write specifications using it. Generic

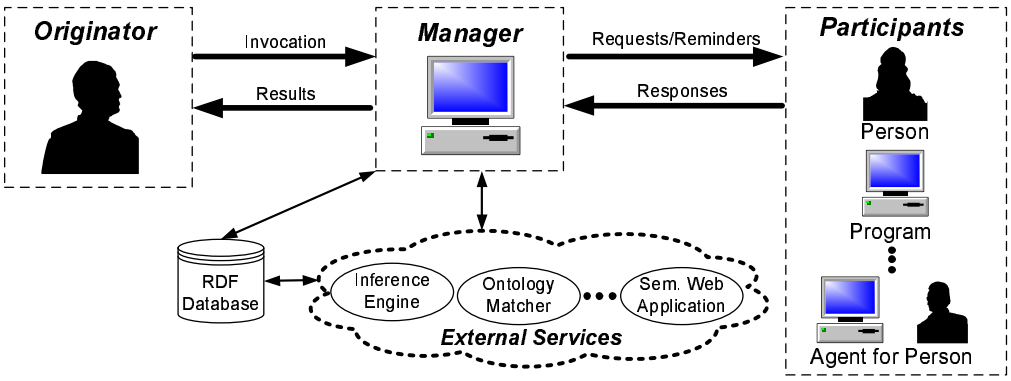

Fig. 1. The invocation and execution of a SEP. The originator is typically a person, but also could be an automated program. The originator invokes a SEP via a simple web interface, and thus need not be trained in the details of SEPs or even understand RDF.

SEPs are written by trained *authors* (who create simple constraints or utility functions to represent the goal of a process) and *invoked* by untrained users. The semantic email system then coordinates the process to provide the formal guarantees we describe later.

Figure 1 illustrates the three primary components of a SEP:

- **Originator:** A SEP is initiated by the *originator*, who is typically a person, but could be an automated program or agent.
- **Manager:** The originator invokes a new SEP by sending a message to the semantic email *manager*. The manager sends email messages to the *participants*, handles responses, and requests changes as necessary to meet the originator's goals. The manager stores all data related to the process in an RDF supporting data set, which may be configured to allow queries by external services (or other managers). To accomplish its tasks, the manager may also utilize external services such as inference engines, ontology matchers, and other Semantic Web applications, as described further below. The manager may be a shared server or a program run directly by the originator.
- **Participants:** The participants respond to messages received about the process. A participant may be a person, a standalone program (e.g., to represent a resource such as a conference room), or a software agent that acts on behalf of a person (e.g., to automatically respond to requests where possible, deferring others to the person). We assume that email addresses uniquely determine individuals or sets of potential participants in the process.

Informally speaking, the execution of a process is achieved by the supporting data set and the set of data updates that email recipients make as they respond. In the model we describe, data is represented as a set of relations (i.e., the relational database model). However, as the application domains get more complex, we expect to use a richer representation language. To enable these future extensions as well as interactions with other Semantic Web applications, our system implements this representation using the Jena RDF storage system [37].

We illustrate our formalization with the running example of a "balanced potluck." The originator invokes a process to announce the potluck and ask everyone whether they are bringing an appetizer, entree, or dessert. The originator also expresses a set of goals for the potluck. For example, he may specify that the number of appetizers, entrees, or desserts should differ by at most two. Note that, while this particular problem has a number of other uses (e.g., distributing N persons evenly among K committees or time slots), it is just an example. Both our formalization and implementation of SEPs support a much broader range of uses.

The manager seeks to expedite the execution of this process and to achieve the originator's goals. There are a number of ways in which reasoning can enhance the manager's operation:

- **Predicting responses:** The manager may be able to infer the likely response of some participants even before sending any requests. For instance, the manager could employ another semantic web application or data source to detect that a suggested meeting time is unacceptable for a certain participant, based on information from calendars, course schedules, or other processes. The manager could use this information either to warn the originator as the process is being created, or to serve as a surrogate response until a definitive answer is received. Also, the manager could add a helpful annotation to the request sent to the participant, indicating what time is likely to be a conflict and why. As suggested above, this same reasoning could also be profitably employed on the participant's end, where an agent may have additional information about the participant's schedule.
- **Interpreting responses:** Typically, the originator will provide the participants with a finite set of choices (e.g., `Appetizer, Entree, Dessert`). However, suitable reasoning could enable substantially more flexibility. For instance, we could allow a potluck participant to respond with any value (either in plain text or in some formal language). Then, the manager could use a combination of information extraction or wrapper techniques (e.g., [15,31]) and/or ontology matching algorithms (e.g., [14,13]) to map the participant's response into the potluck's ontology. There are several interesting outcomes to this mapping. First, the response may directly map to one of the original potluck choices (e.g., "Cake" is an instance of `Dessert`). Second, the response may map to multiple choices in our ontology (e.g., "Jello salad" may be both an `Appetizer` and a `Dessert`). In this case, the manager might consider the response to be half of an appetizer and a dessert, or postpone the decision to a later time and classify it as is most convenient. [2] Third, the response may not map to any given choice, but may still be a subclass of `Food` (e.g., a "Sorbet" is a `Palette Cleanser`); here the manager might accept the response but exclude it from the goal calculations. Fourth, the response

[2] This a very simple form of semantic negotiation; more complex techniques could also be useful [55].

may map to a known ontology element that is not `Food` (e.g., "A hat"). Finally, the response may not map to any known element. In these latter two cases, the manager may either reject the response or notify the originator.

- **Recommending interventions:** Reasoning can also assist the manager with directing the process towards outcomes consistent with the originator's goals. For instance, if the manager detects that a potluck process is becoming unbalanced, it could refuse to accept certain responses, request changes from some participants, or warn the originator that further action is needed. In this case reasoning is needed to deduce the likely outcome of a process from the current state, and the likely effects of possible interventions.

In this work we focus on using reasoning for recommending interventions, leaving the other two items for future work. Specifically, we provide two different approaches for modeling the originator's goals and when to intervene. In the logical model (Sections 3 and 4), the originator specifies a set of *constraints* over the data set that should be satisfied by any process outcome, while in the decision-theoretic model (Section 5) the originator provides a function representing the *utility* of possible process outcomes. Below we consider each in turn, discuss possible variants, and present results for fundamental reasoning tasks that can determine how and when the manager should intervene.

3 Logical Model of SEPs

We now introduce our model of a logical semantic email process (L-SEP) and analyze important inference problems for this model.

3.1 Definition of L-SEPs

A L-SEP is a 5-tuple $\Lambda(P, D, R, M, C_D)$ with parts as follows:

Participants P: the set of participants in the process. Note that P may include the originator.

Supporting data set D: the set of relations that holds all data related to the process. The initial contents of D are specified by the originator (usually to be a set of default values for the columns). With each relation in D we associate a schema that includes:

- a relation name and names, data types, and range constraints for the attributes. A special data type is emailAddress, whose values are elements of the set P. Attributes may have default values.

- possibly a distinguished from attribute, of type emailAddress, which means that rows in the relation whose from value is p can only result from messages from the participant p. The from attribute may be declared unique, in which case every participant can only affect a single row in the table.

Responses R: the set of possible responses to the originator's email. R is specified as follows:

- Attributes: the set of attributes in D that are affected by responses from participants. This set of attributes *cannot* include any from attributes.
- Insert or Update: a parameter specifying whether participants can only add tuples, only modify tuples, or both. Recall that if there is a from field then all changes from p pertain only to a particular set of tuples.
- Single or Many: a parameter specifying whether participants can send a single response or more than one. As we explain in the next section, some responses may be *rejected* by the system. By single, we mean one non-rejected message.

Messages M: the set of messages that the manager may use to direct the process, e.g., to remind the participants to respond or to *reject* a participant's response.

Constraints C_D: the set of constraints for every relation in D. These constraints C_D are specified in a language that includes conjunction and disjunction of atomic predicates. Atomic predicates compare two terms, or a term with a set. We allow comparison predicates ($=, \neq, <, \leq$), LIKE, and $\in, \notin$ between a constant and an enumerated finite set. A term may be a constant, an attribute variable (the value of a specific attribute in a row), an expression combining two terms with any arithmetic operator, or an aggregate applied to a column of a relation (or to a subset of the rows that satisfy an (in)equality predicate).

Example: In our example, D contains one table named Potluck with two columns: email, a from attribute of type `emailAddress` and declared to be unique, and bringing, with the range constraint Potluck.bringing $\in$ { `not-coming, appetizer, entree, dessert, NULL` }. The set of possible responses R is { `not-coming, appetizer, entree, dessert` }. In addition, C_D contains a few constraint formulas similar to the abstract one below, specifying that the potluck should be balanced:

(COUNT(*) WHERE bringing = 'dessert') $\leq$
(COUNT(*) WHERE bringing = 'appetizer') + 2

Finally, the set of messages in our example includes (1) the initial message announcing the potluck and asking what each person is bringing, (2) messages informing each responder whether their response was accepted or not, (3) a reminder to those who have not responded 2 days before the potluck, (4) regular messages

to the originator reporting the status of the responses, and (5) a message to the originator in the event that everyone has responded.

3.2 Inference for L-SEPs

Given the formal model for an L-SEP we can now pose a wide variety of inference problems, whose results can serve to assist in the manager's operation. This section describes the first such inference problem, which has different variations.

The core problem we want to address is determining whether an L-SEP will terminate in an *acceptable* state, i.e., a state that satisfies C_D. The input to the inference problem includes the constraints C_D and possibly the *current* state of D along with a response r from a participant. The output of the inference problem is a condition that we will check on D and r to determine whether to *accept* r. In our discussion, we assume that r is a legal response, i.e., the values it inserts into D satisfy the range constraints on the columns of D; if not, the manager can respond with error messages until a legal response is received. Our goal is to automatically determine whether to *accept* r given the current state and C_D.

The space of possible inference problems is defined by several dimensions:

- **Necessity vs. possibility:** As in modal logics for reasoning about future states of a system [48,18], one can either look for conditions that guarantee that *any* sequence of responses ends in a desired state (the $\Box$ operator), or that it is possible that *some* sequence ends in a desired state (the $\Diamond$ operator).
- **Assumptions about the participants:** In addition to assuming that all responses are legal, we can consider other assumptions, such as: (1) *all* the participants will respond to the message or (2) the participants are flexible, i.e., if asked to change their response, they will cooperate.
- **The type of output condition:** At one extreme, we may want a constraint C_r that the manager checks on D when a response r arrives, where C_r is specified in the same language used to specify C_D. At another extreme, we may produce an arbitrary procedure with inputs D and r that determines whether to accept r. We note that a constraint C_r will inevitably be weaker than an arbitrary algorithm, because it can only inspect the state of D in very particular ways. As intermediate points, we may consider constraints C_r in more expressive constraint languages. Note that in cases where we can successfully derive C_r, we can use database triggers to implement modifications to D or to indicate that r should be rejected.

As a very simple example, consider the case where we want *all* response sequences to end in an acceptable state, we make no assumptions on the participants except that we can elicit a legal response from them, and we are interested in deriving a constraint C_r that will be checked when a response arrives. If the initial state of D is an acceptable state, then simply setting C_r to be C_D provides a sufficient condition;

i.e., we only let the data set D be in states that satisfy C_D. In the example of the balanced potluck, we would not accept a response with a dessert if that would lead to having 3 more desserts than entrees or appetizers. For a giveaway process, we would not accept a request that caused the total number of tickets claimed to be more than the number that is available.

In many cases, such a conservative strategy will be overly restrictive. For example, we may want to continue accepting desserts so long as it is still *possible* to achieve a balanced potluck. Furthermore, this approach is usable only when the constraints are initially satisfied, even before any responses are received, and thus greatly limits the types of goals that can be expressed. This leads us to the following inference problem.

3.3 Ultimate Satisfiability

We now describe our central result concerning inference for L-SEPs. Our goal is to find necessary and sufficient conditions for accepting a response from a participant. To do that, we cut across the above dimensions as follows. Suppose we are given the data set D after 0 or more responses have been accepted, and a new response r. Note that D does not necessarily satisfy C_D, either before or after accepting r. The manager will accept r if it is *possible* that it will lead to a state satisfying C_D (i.e., considering the $\Diamond$ temporal operator). We do not require that the acceptance condition be expressed in our constraint language, but we are concerned about whether it can be efficiently verified on D and r. We assume that D defines some constant number of attributes (e.g., `emailAddress, bringing`). Furthermore, we assume that participants can only update their (single) row, and only do so once.

Definition 3.1 (**ultimate satisfiability**) Given a data set D, a set of constraints C_D on D, and a response $r \in R$, we say that D is ultimately satisfiable w.r.t. r if there exists a sequence of responses from the participants, beginning with r, that will put D in a state that satisfies C_D. □

In what follows, let C be our constraint language where we allow both conjunction and disjunction of atomic predicates. A term in a predicate of C_D may select a group of rows in an attribute A, and aggregate the value of the corresponding values in an attribute B. We consider the aggregation functions COUNT, MIN, MAX, SUM, and AVERAGE. In addition, we define the following:

Definition 3.2 (**bounded constraints**) Given a data set D and a set of constraints C_D on D, we say that C_D is bounded iff one of the following holds:

- **Domain-bounded:** the predicates of C_D only refer to attributes whose domain size is at most some constant L.
- **Constant-bounded:** the predicates of C_D refer to at most K distinct constants, and the only aggregate used by C_D is COUNT. □

All of the examples in this paper may be described by constraints that satisfy the constant-bounded, COUNT-only condition above, while the domain-bounded case may be useful for SEPs that require more complex interactions. Using this definition, we can show that ultimate satisfiability is difficult in general, but much more tractable if the constraints are bounded:

Theorem 1 *Let Λ be an* L-SEP *with N participants and constraints C_D. If C_D may be any set of constraints permitted by the language C, then ultimate satisfiability is NP-complete in N. If C_D is bounded, then determining ultimate satisfiability is polynomial time in N and $|C_D|$.*

As an example of applying this theorem to the balanced potluck, suppose a new dessert response arrives. At that point, the inference procedure will (1) determine the maximal number of people who *may* come to the potluck (i.e., the number of participants minus the number of people who replied `not-coming`), (2) check that even if the dessert response is accepted, then there are still enough people who have not answered such that the ultimate set of dishes could be balanced. Similar reasoning applies to other processes, e.g., to ensure that at least one person will sign up for each spot in a colloquium series.

The theorem is proved by enumerating representative states of the data set, each of which corresponds to a number of potential states that are all equivalent with respect to the constraints. The key is to express the constraints in terms of variables representing *aggregates* on the number of participants with each response. See the appendices for the complete proof.

In comparison to related work, the challenge here is reasoning about the possible relationships between *aggregate* values (current and future), given a particular state of D. Reasoning about aggregation has received significant attention in the query optimization literature [50,33,11,20] and some in the description logic literature (e.g., [3]). This body of work considered the problem of optimizing queries with aggregation by moving predicates across query blocks, and reasoning about query containment and satisfiability for queries involving grouping and aggregation. In contrast, our result involves considering the current state of the database to determine whether it can be brought into a state that satisfies a set of constraints. Furthermore, since C_D may involve several grouping columns and aggregations, they cannot be translated into single-block SQL queries, and hence the containment algorithms will not carry over to our context.

To the best of our knowledge, formalisms for reasoning about workflow [44,43] or about temporal properties of necessity and possibility have not considered reasoning about aggregation. For instance, Abiteboul et al. [1] define a notion of goal reachability for *relational transducers* that is similar to our ultimate satisfiability (see also [22] for extensions to this model and a survey of other related work). Various restrictions on the model allow decidability of goal reachability in P, NP, or NEXPTIME, but none of these restrictions permit goals involving aggre-

gation. Likewise, workflow formalisms have generally been restricted to reasoning about temporal and causality constraints. Such formalisms could potentially convert aggregation constraints to temporal constraints by enumerating all possible data combinations, but this may result in an exponential number of states. One exception is the recent work of Senkul et al. [52], who extend workflows to include *resource constraints* based on aggregation. Each such constraint, however, is restricted to performing a single aggregation with no grouping (and thus could not express the potluck constraint given in the earlier example). In addition, their solution is based on general constraint solving and thus will take exponential time in the worst case. We have shown, however, that in our domain L-SEPs can easily express more complex aggregation constraints while maintaining polynomial-time inference complexity for bounded constraints.

4 Explanation Generation for L-SEPs

While executing, an L-SEP utilizes rejections to influence the eventual outcome. However, the success of these interventions depends on the extent to which they are understood by the participants. For instance, the rejection "Sorry, the only dates left are May 7 and May 14" is much more likely to elicit cooperation from a participant in a seminar scheduling SEP than the simpler rejection "Sorry, try again." For a particular set of constraints, the author of a SEP could manually specify how to create such explanations, but this task can be very difficult when constraints interact or depend on considering possible future responses. Thus, below we consider techniques for automatically generating explanations based on *what* responses are acceptable now and *why* the participant's original response was not acceptable.

We begin by defining more precisely a number of relevant terms. Given an L-SEP, the *current state* D is the state of the supporting data set given all of the responses that have been received so far. We assume that the number of participants is known and that each will eventually respond. Following the earlier discussion regarding necessity vs. possibility, we allow constraint satisfaction to be defined in two different ways:

Definition 4.1 (**MustConstraint**) A `MustConstraint` C is a constraint that is satisfied in state D iff evaluating C over D yields `True`. □

Definition 4.2 (**PossiblyConstraint**) A `PossiblyConstraint` C is a constraint that is *ultimately satisfiable* in state D if there exists a sequence of responses from the remaining participants that leads to a state D' so that evaluating C over D' yields `True`. □

For simplicity, we assume that the constraints C_D are either all `MustConstraints` or all `PossiblyConstraints`, though our results for `PossiblyConstraints` also hold when C_D contains both types.

4.1 Acceptable Responses

Often the most practical information to provide to a participant whose response led to an intervention is the set of responses that would be "acceptable" (e.g., "An Appetizer or Dessert would be welcome" or "Sorry, I can only accept requests for 2 tickets or fewer now"). This section briefly considers how to calculate this *acceptable set*.

Definition 4.3 (**acceptable set**) Let Λ be an L-SEP with current state D and constraints C_D on D. Then, the *acceptable set* A of Λ is the set of legal responses r such that D would still be satisfiable w.r.t. C_D after accepting r. □

For a `MustConstraint`, this satisfiability testing is easy to do and we can compute the acceptable set by testing some small set of representative responses. For a `PossiblyConstraint`, the situation is more complex:

Theorem 2 *Let Λ be an* L-SEP *with N participants and current state D. If the constraints C_D may be any set of constraints permitted by the language C, then computing the acceptable set A of Λ is NP-hard in N. If C_D is bounded, then this problem is polynomial time in N, $|A|$, and $|C_D|$.*

In this case we can again compute the acceptable set by testing satisfiability over some small set of representative values; this testing is polynomial iff C_D is bounded (Theorem 1). In addition, if we represent A via a set of ranges of acceptable values, instead of explicitly listing every acceptable value, then the total time is polynomial in only N and $|C_D|$.

4.2 Explaining Interventions

In some cases, the acceptable set alone may not be enough to construct a useful explanation. For instance, suppose an L-SEP invites 4 professors and 20 students to a meeting that at least 3 professors and a quorum of 10 persons (professors or students) must attend. When requesting a change from a professor, explaining *why* the change is needed (e.g., "We need you to reach the required 3 professors") is much more effective than simply informing them *what* response is desired (e.g., "Please change to Yes"). A clear explanation both motivates the request and rules out alternative reasons for the request (e.g., "We need your help reaching quorum") that may be less persuasive (e.g., because many students could also help reach quorum). This section discusses how to generate explanations for an intervention based on identifying the constraint(s) that led to the intervention. We do not discuss the additional problem of translating these constraints into a natural language suitable for sending to a participant, but note that even fairly simple explanations (e.g., "Too many Appetizers (10) vs. Desserts (3)") are much better than no explanation.

Conceptually, an L-SEP decides to reject a response based on constructing a *proof tree* that shows that some response r would prevent constraint satisfaction. However, this proof tree may be much too large and complex to serve as an explanation for a participant. This problem has been investigated before for expert systems [45,54], constraint programming [26], description logic reasoning [40], and more recently in the context of the Semantic Web [41]. These systems assumed proof trees of arbitrary complexity and handled a wide variety of possible deduction steps. To generate useful explanations, key techniques included abstracting multiple steps into one using rewrite rules [40,41], describing how general principles were applied in specific situations [54], and customizing explanations based on previous utterances [10].

In our context, the proof trees have a much simpler structure that we can exploit. In particular, proofs are based only on constraint satisfiability (over one state or all possible future states), and each child node adds one additional response to the parent's state in a very regular way. Consequently, we will be able to summarize the proof tree with a very simple type of explanation. These proof trees are defined as follows:

Definition 4.4 (**proof tree**) Given an L-SEP Λ, current state D, constraints C_D, and a response r, we say that P is a *proof tree* for rejecting r on D iff:

- P is a tree where the root is the initial state D.
- The root has exactly one child D_r, representing the state of D after adding r.
- If C_D is all `MustConstraints`, then D_r is the only non-root node.
- If C_D is all `PossiblyConstraints`, then for every node n that is D_r or one of its descendants, n has all children that can be formed by adding a single additional response to the state of n. Thus, the leaf nodes are only and all those possible final states (e.g., where every participant has responded) reachable from D_r.
- For every leaf node l, evaluating C_D over the state of l yields `False`. □

Figure 2A illustrates a proof tree for `MustConstraints`. Because accepting r leads to a state where some constraint (e.g., C_T) is not satisfied, r must be rejected. Likewise, Figure 2B shows a proof tree for `PossiblyConstraints`, where C_P and C_Q represent the professor and quorum constraints from the example described above. Since we are trying to prove that there is no way for the constraints to be ultimately satisfied (by any outcome), this tree must be fully expanded. For this tree, every leaf (final outcome) does not satisfy some constraint, so r must be rejected.

We now define a simpler explanation based upon the proof tree:

Definition 4.5 (**sufficient explanation**) Given an L-SEP Λ, current state D, constraints C_D, and a response r such that a proof tree P exists for rejecting r on D, then we say that E is a *sufficient explanation* for rejecting r iff,

- E is a conjunction of constraints that appear in C_D, and
- for every leaf node l in P, evaluating E over the state of l yields `False`. □

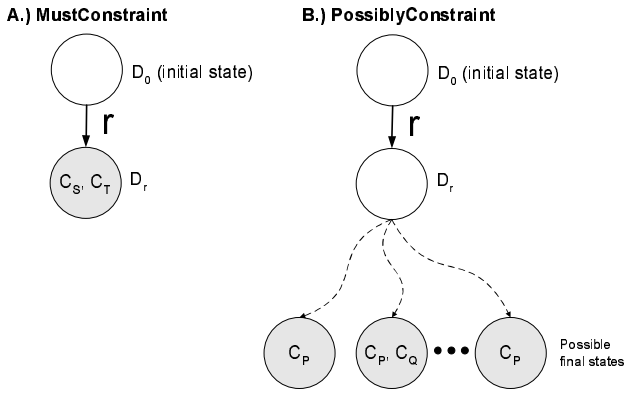

Fig. 2. Examples of proof trees for rejecting response r. Each node is a possible state of the data set, and node labels are constraints that are *not* satisfied in that state. In both cases, response r must be rejected because every leaf node (shaded above) does not satisfy some constraint.

Intuitively, a sufficient explanation E justifies rejecting r because E covers every leaf node in the proof tree, and thus precludes ever satisfying C_D. Note that while the proof tree for rejecting r is unique (modulo the ordering of child nodes), an explanation is not. For instance, an explanation based on Figure 2A could be C_S, C_T, or $C_S \wedge C_T$. Likewise, a valid explanation for Figure 2B is $C_P \wedge C_Q$ (e.g., no way satisfy both the professor and quorum constraints) but a more precise explanation is just C_P (e.g., no way to satisfy the professor constraint). The smaller explanation is often more compelling, as we argued for the meeting example, and thus to be preferred [12]. In general, we wish to find an explanation of minimum size (i.e., with the fewest conjuncts):

Theorem 3 *Given an* L-SEP Λ *with* N *participants, current state* D*, constraints* C_D*, and a response* r*, if* C_D *consists of* `MustConstraints`*, then finding a minimum sufficient explanation* E *for rejecting* r *is polynomial time in* N *and* $|C_D|$*. If* C_D *consists of* `PossiblyConstraints`*, then this problem is NP-hard in* N *and NP-hard in* $|C_D|$*.*

Thus, computing a minimum explanation is feasible for `MustConstraints` but likely to be intractable for `PossiblyConstraints`. For the latter, the difficulty arises from two sources. First, checking if any particular E is a sufficient explanation is NP-hard in N (based on a reduction from ultimate satisfiability); this makes scaling SEPs to large numbers of participants difficult. Second, finding a minimum such explanation is NP-hard in the number of constraints (by reduction from SET-COVER [23]); this makes explanation generation for complex goals challenging. Fortunately, in many common cases we can simplify this problem to permit a polynomial time solution:

Theorem 4 *Given an* L-SEP Λ *with* N *participants, current state* D*, constraints* C_D*, and a response* r*, if* C_D *is bounded and the size of a minimum explanation is no more than some constant* J*, then computing a minimum explanation* E *is polynomial time in* N *and* $|C_D|$*.*

This theorem holds because a candidate explanation E can be checked in polynomial time when the constraints are bounded, and restricting E to at most size J means that the total number of explanations that must be considered is polynomial in the number of constraints. Both of these restrictions are quite reasonable. As previously mentioned, bounded constraints permit a wide range of functionality. Likewise, SEP explanations are most useful to the participants when they contain only a small number of constraints, and this is adequate for many SEPs (as in the meeting example above). If no sufficient explanation of size J exists, the system could either choose the best explanation of size J (to maintain a simple explanation), approximate the minimum explanation with a greedy algorithm, or fall back on just providing the participant with the acceptable set described in the previous section.

5 Decision-theoretic model

The logical model of SEPs described above supports a number of useful inferences that have both theoretical and practical applications. This model, however, has a number of shortcomings. First, L-SEPs, like logical theories in general, make no distinctions among unsatisfied outcomes. In our example, there is no way for L-SEPs to strive for a "nearly-balanced" potluck, since all unbalanced potlucks are equivalently undesirable. Second, an L-SEP ignores the cost of the actions taken in pursuit of its goals. For instance, a potluck L-SEP will always reject a response that results in unsatisfiable constraints, even if rejecting that response (e.g., from an important official) may produce far worse effects than a slightly unbalanced potluck. Finally, L-SEPs make a very strong assumption that participants are always willing to change their responses if rejected. For instance, participants in a meeting scheduling process may be somewhat accommodating, but may refuse to modify a rejected response due to other commitments.

To address these limitations, we offer a decision-theoretic approach. We describe the goal of a decision-theoretic SEP (D-SEP) by a utility function over the outcome of the process that takes into consideration the cost of all actions required to achieve that outcome. In addition, instead of *rejecting* responses, the decision-theoretic model sometimes *suggests* that participants modify their choices. For instance, the balanced potluck uses a utility function that measures the extent to which the final meal selection is balanced, minus the costs (social or otherwise) of asking some participants to switch their responses. Below we formalize this model and then examine the tractability of finding optimal policies for it.

5.1 Definition of D-SEPs

A decision-theoretic SEP is a 6-tuple, $\delta(P, S, V, A, U, T)$. Note that the first five components of this tuple correspond roughly to the five components in our model for L-SEPs.

- **Participants** P**:** the set of participants, of size N.
- **States** S**:** the set of possible states of the system. The state describes both the current responses received and the outgoing change requests sent by the system.
- **Values** V**:** the set of possible values for participants to choose from (e.g., $V = \{appetizer, entree, dessert\}$).
- **Actions** A**:** the set of actions available to the system after sending out the initial message. Actions we consider are $NoOp$ (do nothing until the next message arrives), SW_v (ask a participant to switch their response from v to something else), or $Halt$ (enter a terminal state, typically only permitted when a message has been received from every participant). Other variants of actions are also useful (e.g., ask a participant to switch from v to a particular value w); such additions do not fundamentally change the model or our complexity results.
- **Utilities** $U(s, a)$**:** the utility from executing action a in state s. For the potluck example, $U(s, SW_v)$ is the (negative) utility from making a change suggestion, while $U(s, Halt)$ is the utility based on the final potluck balance.
- **Transitions** $T(s, a, s')$**:** the probability that the system will transition to state s' after performing action a in state s. However, rather than having to specify a probability for each transition, these are computed from a smaller set of building blocks. For instance, ρ_v is the probability that a participant will originally respond with the value v; ρ_{vw} is the probability that, when asked to switch from the choice v, a participant will change their response to w (ρ_{vv} is the probability that a participant refuses to switch).

The execution of the process proceeds in discrete steps, where at each step the manager decides upon an action to take (possibly $NoOp$). The outcome of this action, however, is uncertain since the manager is never sure of how participants will respond. The transition function $T()$ models this uncertainty.

A *policy* π describes what action the manager will take in any state, while $\pi(s)$ denotes the action that the manager will take in a particular state s. An *optimal policy* $\pi^\star$ is a policy that maximizes the expected utility $U(\delta)$ of the process, where

$$U(\delta) = U(s_1, a_1) + U(s_2, a_2) + ... + U(s_j, a_j)$$

for the sequence of states and actions $\{(s_1, a_1), ..., (s_j, Halt)\}$.

D-SEPs are a special case of *Markov Decision Processes* (MDPs), a well-studied formalism for situations where the outcome of performing an action is governed by a stochastic function and costs are associated with state transitions [49]. Conse-

quently, we could find the optimal policy for a D-SEP by converting it to an MDP and using known MDP policy solvers. [3] However, this would not exploit the special characteristics of D-SEPs that permit more efficient solutions, which we consider below.

5.2 Variations of D-SEPs

As with our logical model, the space of possible D-SEPs is defined by several dimensions:

- **Restrictions on making suggestions:** Most generally, the manager may be allowed to suggest changes to the participants at *any time*, and to do so repeatedly. To be more user-friendly, we may allow the manager to make suggestions anytime, but *only once* per participant. Alternatively, if users may be expected to make additional commitments soon after sending their response (e.g., purchasing ingredients for their selected dish), then we may require the manager to respond with any suggestion *immediately* after receiving a message, before any additional messages are processed.
- **Assumptions about the participants:** In addition to the assumed probabilities governing participant behavior, we may also wish to assume that all participants will eventually respond to each message they receive. Furthermore, we might assume that participants will respond *immediately* to any suggestions that they receive (particularly if the manager also responds immediately to their original message), or instead that they can respond to suggestions *anytime*.
- **The type of utility functions:** At one extreme, we might allow complex utility functions based upon the individual responses of the participants (e.g., "+97 if Jay is bringing dessert"). Often, however, such precision is unnecessary. For instance, all potluck outcomes with 8 desserts and 1 entree have the same low utility, regardless of who is bringing what dish.

Below we consider the impact of these variations on the complexity of computing the optimal policy.

5.3 Computing the Optimal Policy

In this section we examine the time complexity of computing the optimal policy $\pi^\star$ for a D-SEP. We begin by considering a D-SEP with an arbitrary utility function and then examine how restrictions to the utility function and the permitted quantity

[3] Specifically, D-SEPs are "Stochastic Shortest-Path" MDPs where the terminal state is reachable from every state, so an optimal policy is guaranteed to exist [5]. Incorporating additional features from *temporal MDPs* [9] would enable a richer model for D-SEPs (e.g., scheduling a meeting should be completed before the day of the meeting). However, existing solution techniques for TMDPs do not scale to the number of participants required for semantic email.

and timing of suggestions make computing $\pi^\star$ more tractable. In all cases we assume that the participants will eventually respond to each message and suggestion that they receive. (We can relax this assumption by representing in the model the probability that a participant will *not* respond to a message.) The following theorem is proved by reduction from QBF (quantified boolean formula) and the EXPTIME-hard game G_4 [53,34]:

Theorem 5 *Let δ be a* D-SEP *with N participants where the utility $U(s, a)$ is any deterministic function over the state s and the current action a. If the manager can send only a bounded number of suggestions to each participant, then determining $\pi^\star$ is PSPACE-hard in N. If the manager can send an unlimited number of suggestions, then this problem is EXPTIME-hard in N. The corresponding problems of determining if the expected utility of $\pi^\star$ for δ exceeds some constant θ are PSPACE-complete and EXPTIME-complete, respectively.*□

Thus, for the case of arbitrary utility functions determining $\pi^\star$ for a D-SEP is impractical for large values of N. (Conversion to an MDP offers little help, since the MDP would require a number of states exponential in N.) Note that this is a significant limitation, since for many D-SEPs it is natural to wish to scale to large numbers of participants (e.g., for large meetings or company-wide surveys). Below, we begin to make the calculation of $\pi^\star$ more tractable by restricting the type of utility function:

Definition 5.1 (**K-Partitionable**) The utility function $U(s, a)$ of a D-SEP is *K-partitionable* if it can be expressed solely in terms of the variables $a, C_1, ..., C_K$ where a is the current action chosen by the manager and each C_i is the number of participants who have responded with value V_i in state s. □

Intuitively, a utility function is K-partitionable if what matters is the number of participants that belong to each of a fixed number of K groups, rather than the specific participants in each of these groups. For instance, the utility function of our example potluck is *4-partitionable*, because all that matters for evaluating current and future utilities is the current number of participants that have responded `Appetizer, Entree, Dessert,` and `Not-Coming`. In this case a simple utility function might be:

$$U(s, Halt) = -\alpha(|C_A - C_E|^2 + |C_A - C_D|^2 + |C_E - C_D|^2)$$
$$U(s, SW_v) = -1$$

where α is a scaling constant and C_A, C_E, and C_D are the numbers of appetizers, entrees, and desserts, respectively. Note that the maximum utility here is zero.

A K-partitionable utility function is analogous to the COUNT-only constraint language of Theorem 1. As with Theorem 1, we could allow more complex utility functions (e.g., with variables representing the MAX, SUM, etc. of the underlying

responses); with suitable restrictions, such functions yield polynomial time results similar to those described below. Note, however, that the simpler K-partitionable definition is still flexible enough to support all of the SEPs discussed in this paper. In particular, a K-partitionable utility function may still distinguish among different types of people by counting responses differently based on some division of the participants. This technique increases the effective value of K, but only by a constant factor. For instance, the utility function for a meeting scheduling process that desires to have the number of *faculty* members attending ($C_{yes,F}$) be at least three and the number of *students* attending ($C_{yes,S}$) be as close as possible to five, while strongly avoiding asking faculty members to switch, might be:

$$\begin{aligned} U(s, Halt) &= -\alpha[max(3 - C_{yes,F}, 0)]^2 - \beta|C_{yes,S} - 5|^2 \\ U(s, SW_{no,F}) &= -10 \\ U(s, SW_{no,S}) &= -1 \end{aligned}$$

A D-SEP that may make an unlimited number of suggestions but that has a *K-partitionable* utility function can be represented as an "infinite-horizon" MDP with just $O(N^{2K})$ reachable states. Consequently, the D-SEP may be solved in time polynomial in N with the use of linear programming (LP), though alternative methods (e.g., policy iteration, simplex-based LP solvers) that do not guarantee polynomial time may actually be faster in practice due to the large polynomial degree of the former approach [35].

Furthermore, if we also restrict the system to send only one suggestion to any participant (likely a desirable property in any case), then computing the optimal policy becomes even more tractable:

Theorem 6 *Let δ be a* D-SEP *with N participants where $U(s, a)$ is K-partitionable for some constant K and where the system is permitted to send at most one suggestion to any participant. Then $\pi^\star$ for δ can be determined in $O(N^{3K})$ time. (If the system can send at most L suggestions to any participant, then the total time needed is $O(N^{(2L+1)K})$.)* □

Table 1 summarizes the results presented above as well as a few other interesting cases ("Immediate" and "Synchronous"). These results rely on two key optimizations. First, we can dramatically reduce the number of distinct states via K-partitioning [4]; this permits $\pi^\star$ to be found in polynomial time. Second, we can ensure that the state transition graph is acyclic (a useful property for MDPs also noted in other contexts [6]) by bounding the number of suggestions sent to each participant; this enables us to find $\pi^\star$ with simple graph search algorithms instead of with policy iteration or linear programming. Furthermore, this approach enables the use of existing heuristic search algorithms where an exact computation remains

[4] See Boutilier et al. [8] for an alternative (though not guaranteed) use of domain structure to reduce the effective number of states.

Restrictions	**Description of Restrictions**	**Complexity with arbitrary utility function**	**Complexity when *K-partitionable***
AnyUnlimited	Manager may suggest changes at any time, and may send an unlimited number of suggestions to any participant.	EXPTIME-hard	MDP with $O(N^{2K})$ *states*
AnyOnce	Manager may suggest changes at any time, but only once per participant.	PSPACE-hard	$O(N^{3K})$ time
Immediate	Manager may suggest changes only immediately after receiving a response, once per participant.	PSPACE-hard	$O(N^{2K})$ time
Synchronous	Same as "Immediate", but each participant is assumed to respond to any suggestion before the manager receives any other message.	PSPACE-hard	$O(N^{K})$ time

Table 1
Summary of theoretical results for D-SEPs. The last two columns show the time complexity of finding the optimal policy for a D-SEP with N participants. In general, this problem is EXPTIME-hard but if the utility function is *K-partitionable* then the problem is polynomial time in N. (An MDP can be solved in time guaranteed to be polynomial in the number of states, though the polynomial has high degree). Adding restrictions on how often the manager may send suggestions makes the problem even more tractable. Note that the size of the optimal policy is finite and must be computed only once, even though the execution of a SEP may be infinite (e.g., with "AnyUnlimited").

infeasible. Consequently, with appropriate restrictions many useful D-SEPs can be efficiently solved in polynomial time.

5.4 Explanation Generation for D-SEP*s*

As with L-SEPs, we would like to be able to automatically generate explanations for the manager's interventions. Below we briefly consider this problem in the context of D-SEPs.

Compared to L-SEPs, it is more difficult for a D-SEP to single out specific terms that are responsible for a manager's suggestion, because every term contributes to the process utility to some extent, either positively or negatively. Note, though, that if the manager decides to make a suggestion, then the expected improvement must outweigh the certain cost of this action. Thus, for non-zero costs, there must be a significant difference in the utility of the state where the manager requested a switch (S_{sw}) vs. where the manager did not (S_0).

We seek to identify the terms that explain most of this difference. In particular, given a n-term utility function

$$U(s) = u_1(s) + ... + u_n(s)$$

we define the change δ_u in each utility term as

$$\delta_u = u(S_{sw}) - u(S_0).$$

We wish to identify a set $E \subseteq \{u_1, .., ..u_n\}$ such that:

$$\sum_{u \in E} \delta_u \geq \beta[U(S_{sw}) - U(S_0)]$$

i.e., so that the terms in E explain at least β of the change. Note that, if we can compute the optimal policy in polynomial time, then we can also compute each δ_u in polynomial time.

When generating an explanation, we are primarily interested in terms indicating that a switch is beneficial, i.e., where $\delta_u > 0$. If we only consider such terms, then a greedy algorithm suffices to identify the explanation E of guaranteed minimal size: set E to $\emptyset$, then incrementally add to E the term with the largest δ_u until E explains at least β of the total change. If we wish to consider utility terms with both positive and negative changes, then this problem becomes more challenging (cf., Klein and Shortliffe [29]).

5.5 Discussion

Compared to L-SEPs, the primary advantages of D-SEPs are their ability to balance the utility of the process's goals vs. the cost of additional communication with the participants, and their graceful degradation when goals cannot be completely satisfied. On the other hand, the need to determine suitable utilities and probabilities is an inherent drawback of any decision-theoretic framework. Below we consider techniques to approximate these parameters.

First, the $\pi^\star$ for a D-SEP depends upon the relative value of positive utilities (e.g., having a well-balanced potluck) vs. negative utilities (e.g., the cost of making a suggestion). Our discussion above exhibited a number of simple but reasonable utility functions. In practice, we expect that D-SEPs will provide default utility functions based on their functionality, but would allow users to modify these functions by adjusting parameters or by answering a series of utility elicitation questions [7].

Second, D-SEPs also require probabilistic information about how participants are likely to respond to original requests and suggestions. This information can be determined in a number of ways:

- **User-provided:** The process originator may be able to provide reliable estimates of what responses are likely, based on some outside information or past experience.
- **History-based:** Alternatively, the system itself can estimate probabilities by examining the history of past processes.
- **Dynamically-adjusted:** Instead of or in addition to the above methods, the system could dynamically adjust its probability estimates based on the actual responses received. If the number of participants is large relative to the number of choices, then the system should be able to stabilize its probability estimates well before the majority of responses are received.

Thus, although the need to provide utility and probability estimates is a drawback of D-SEPs compared to L-SEPs, simple techniques can produce reasonable approximations for both. In practice, the choice of whether to use a D-SEP or L-SEP will depend on the target usage and the feasibility of parameter estimation. In our implementation, we allow the originator to make this choice. For D-SEPs, we currently elicit some very basic utility information from the originator (e.g., see Figure 4), and use some probabilities provided by the SEP author for expected participant behavior. Extending our implementation to support history-based and dynamically-adjusted probabilities is future work.

6 Implementation and Usability

We have implemented a complete semantic email system and deployed it in several applications. In doing so, we faced several challenges. This section describes the desiderata for a usable semantic email system, highlights the challenges to achieving these desiderata, and discusses our particular implementation choices.

6.1 Desiderata

To be successful, we argue that any semantic email system (both SEP-based and otherwise) should fulfill the following desiderata:

- **Instant Gratification**: Most importantly, semantic email must provide an immediate, tangible benefit to users. Users must not be expected to annotate outgoing or incoming mail with semantic content for some vague future benefit. Instead, a semantic email system must provide users with existing services that yield immediately obvious benefits. In fact, the notion of instant gratification is key to getting people to invest in structuring their data, and has been the motivation behind our MANGROVE semantic web system [38].

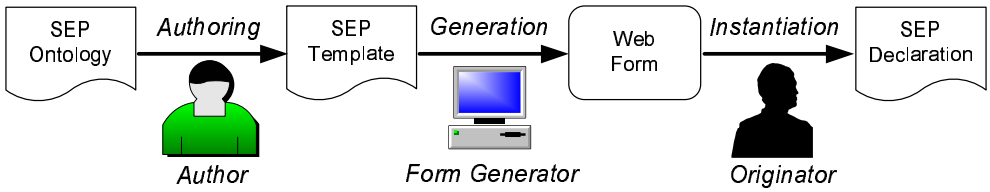

Fig. 3. The creation of a semantic email process (SEP). Initially, an "Author" *authors* a SEP template and this template is used to *generate* an associated web form. Later, this web form is used by the "Originator" to *instantiate* the template. Typically, a template is authored once and then instantiated many times.

- **Gradual Adoption**: At first, semantic email will be initiated by only a small number of "early adopters." If semantic email could be profitably exchanged only among these users, it would have very limited applicability. Thus, to succeed, semantic email must be usable even when some or all of the participants have no experience with or software installed for it.
- **Ease of use**: Semantic email must be simple enough for a non-technical person to use. It should not expect such users to understand RDF, disrupt normal email processing, or require email senders or recipients to use a particular email client.

Below we elaborate on the challenges in implementing a system that achieves these goals.

6.2 Process Creation and Execution

Translating SEP theory to real problems: Applying our SEP theory to real problems requires enabling an originator to easily create an L-SEP or D-SEP model that corresponds to his goals. One option is to build a GUI tool that guides the originator through constructing the appropriate choices, messages, and constraints or utilities for the process. Practically, however, a tool that is general enough to build an arbitrary process is likely to be too complex for untrained users.

Instead, our system is based on the construction of reusable *templates* for specific classes of SEPs. Figure 3 demonstrates this approach. Initially, someone who is assumed to have some knowledge of RDF and semantic email authors a new template using an editor (most likely by modifying an existing template). We call this person the SEP *author*. The template is written in OWL [57] based on a SEP ontology that describes the possible queries, constraints, and messages for a process. For instance, the "balanced potluck" template defines the general balance constraints for the process, but has placeholders for *parameters* such as the participants' addresses, the specific choices to offer, and how much imbalance to permit.

To enable an originator to provide these parameters later, we associate with each template a simple web form that prompts the originator for each parameter. For in-

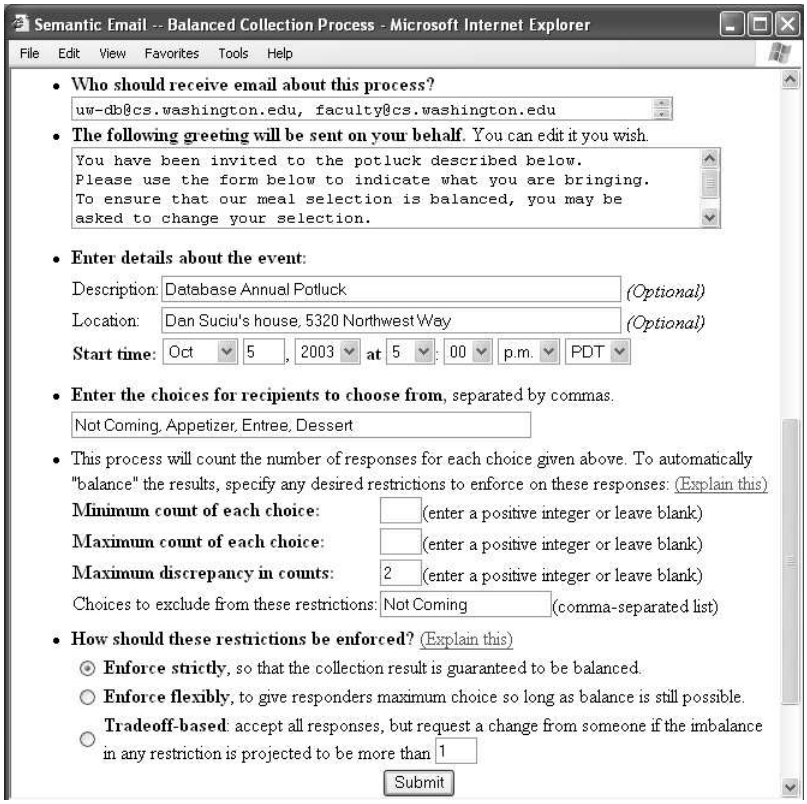

Fig. 4. A web form used to initiate a "balanced collection" process, such as our balanced potluck example. For convenience, clicking submit converts the form to text and sends the result to the server and a copy to the originator. The originator may later initiate a similar process by editing this copy and mailing it directly to the server.

stance, Figure 4 shows such a form for the balanced potluck. Note that the bottom of this form allows users to choose between executing an L-SEP (the "strictly" and "flexibly" options) or a D-SEP (the "tradeoff-based" option). In addition, originators may specify either individuals or mailing lists as participants; for the latter case, the form also asks the originator for an estimate of the total number of people that will respond (not shown in Figure 4).

Our implementation provides a simple tool that can automatically generate such web forms from some additional OWL information in the template. This same tool could also be used to generate a service description for the template, e.g., in WSDL or OWL-S [2]. Then, a program could also serve as an originator by utilizing the service description and template to automatically invoke the process directly.

An untrained originator finds a SEP from a public library of SEP templates and instantiates the template by filling out the corresponding web form, yielding a SEP *declaration* (also in OWL). The originator then invokes the process by forwarding this declaration to the manager. Given the formal declaration, the manager then executes the process, using appropriate L-SEP and D-SEP algorithms to decide how to direct the process via appropriate message rejections and suggestions

Facilitating responses: Another key challenge is enabling participants to respond to messages in a way that is convenient but that can be automatically interpreted by the manager. A number of different solutions are possible:

- **Client software:** We could provide a custom email client that would present the participant with an interface for constructing legal responses, or automatically respond to messages it knows how to handle (e.g., "Decline all invitations for Friday evenings"). This client-based approach, however, requires all participants in a process to install additional software (conflicting with our gradual adoption goal) and is complicated by the variety of mail clients currently in use.
- **Information extraction:** We could allow participants to respond in a natural language (e.g., "I'll bring a dessert"). We could then use wrappers or information extraction techniques to attempt to convert this response to one of the offered choices. This approach is promising but risks having the wrapper fail to extract the correct information.
- **Email or web forms:** We could provide participants with a text-encoded form to fill out, or we could send them a link to a suitable web-based form to use for their response. Embedded HTML forms are also attractive, but unfortunately are not handled uniformly by existing email clients.

While web forms have some advantages, we chose to use email text forms instead because we feel they fit more naturally with how people typically handle incoming messages. In addition, text forms offer a simple solution that works for any participant. Participants respond by replying to the process message and editing the original form. If the manager sends a rejection or suggestion to a participant, the message includes an explanation of the intervention along with a copy of the original form so that the participant can modify their response.

Our earlier discussion generally assumed that participants would send a single acceptable response. However, our implementation does permit participants to "change their mind" by sending additional responses. For the logical model, this response is accepted if changing the participant's original response to the new value still permits the constraints to be satisfied (or if the response must always be accepted, e.g., for `Not-Coming`). For the decision-theoretic model, the new response is always accepted but may lead to a change suggestion based on the modified state of the process.

Manager deployment: Potentially, the manager could be a program run on the originator's personal computer, perhaps as part of his mail client. This permits an easy transition between authoring traditional mails and invoking SEPs, and can also benefit from direct access to the originator's personal information (e.g., calendar, contacts). However, as with providing client software for participants, this approach requires software installation and must deal with the wide variety of existing mail clients.

Our implementation instead deploys the manager as a shared server. The server

SEP name	Procedural approach (number of lines)	Declarative approach (number of lines)	Size Reduction for Declarative
Balanced Potluck	1680	170	90%
First-come, First-served	536	99	82%
Meeting Coordination	743	82	89%
Request Approval	1058	109	90%
Auction	503	98	81%

Table 2
Comparison of the size of a SEP specification in our original procedural prototype [17] (using Java/HTML) vs. in the declarative format described in this paper (using RDF). Overall, the declarative approach is about 80-90% more concise. These values include the HTML/RDF needed for acquiring parameters from the originator.

receives invocations from the originator and sends out an initial message to the participants. Participants reply via mail directly to the server, rather than to the originator, and the originator receives status and summary messages from the server when appropriate. The originator can query or alter the process via additional messages or a web interface.

Discussion: Our server-based approach is easy to implement and meets our gradual adoption and ease-of-use goals since it requires no software installation, works for all email clients, and does not require users (as originators) to read or write RDF. In addition, this method supports our instant gratification goal by providing untrained users with existing, useful SEPs that can be immediately invoked and yield a tangible output (in the form of messages sent and processed on the users' behalf). Finally, we believe that divorcing the processing of semantic email (in the server) from the standard email flow (in the client) will facilitate adoption by ameliorating user concerns about privacy [5] and about placing potentially buggy code in their email client.

In addition, specifying SEP templates and declarations in OWL has a number of advantages. First, unlike the original version of semantic email [17] (which used process-specific procedures), a SEP is described entirely by its OWL declaration. This greatly simplifies the deployment of a new SEP, both because no programming is required and because authors need not run their own server (since shared servers can accept and execute OWL declarations from anyone, something they are unlikely to do for arbitrary code). In addition, these declarations are much simpler and more concise than corresponding specifications written in a procedural language (see Table 2). Furthermore, authoring SEPs in OWL enables the use of a variety of automated tools to ensure that a SEP declaration is valid. Finally, OWL

[5] Only semantic email goes through the server, personal email is untouched. Of course, when the semantic email also contains sensitive information, the security of the server becomes significant.

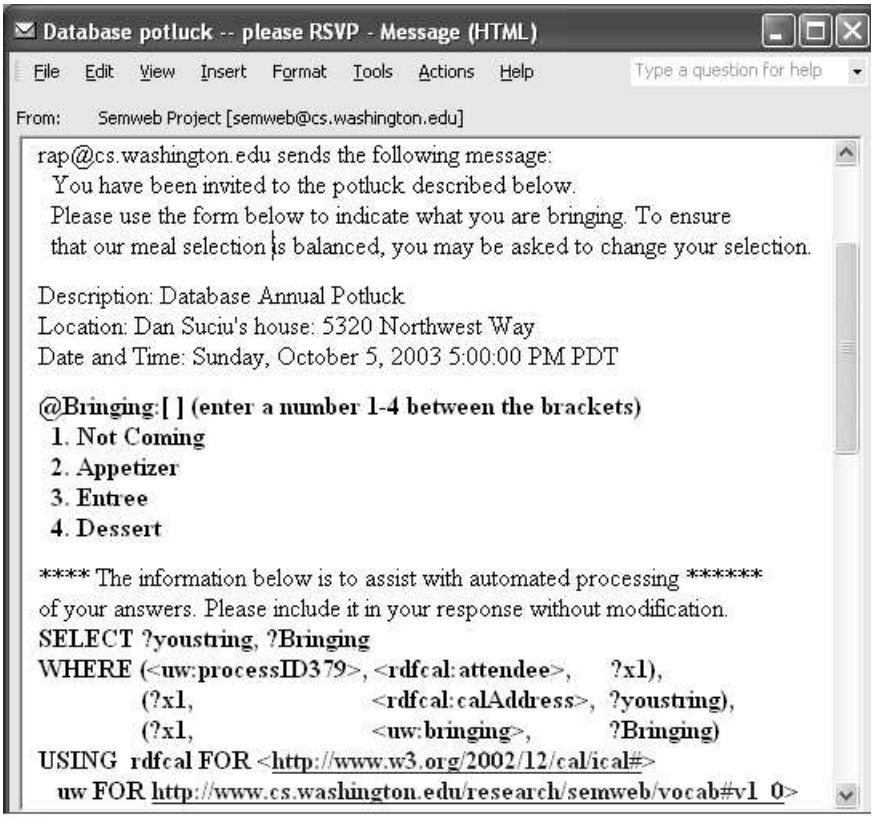

Fig. 5. A message sent to participants in a "balanced potluck" process. The bold text in the middle is a form used for human recipients to respond, while the bold text at the bottom is a RDQL query that maps their textual response to RDF.

specifications could enable future work that automatically composes several SEPs to accomplish more complex goals.

6.3 Human/Machine Interoperability

The previous section highlighted how semantic email messages can be handled by either a human or by a program operating on their behalf. Thus, an important requirement is that every message must contain both a human-understandable portion (e.g., "You're invited to the potluck on Oct 5...") and a corresponding machine-understandable portion. For messages sent to a participant, this approach supports gradual adoption by permitting the originator to send the same message to all participants without any knowledge of their capabilities. For responses, a machine-understandable portion enables the manager to evaluate the message against the process constraints/utilities and take further action. The human-readable component provides a simple record of the response if needed for later review.

In our implementation, we meet this interoperability requirement with a combination of techniques. For responses, a human can fill out the included text form (see Figure 5), which is then converted into RDF at the server with a simple mapping from each field to an unbound variable in a RDQL query associated with the mes-

sage. Alternatively, a machine can respond to the message simply by answering the query in RDF, then applying the inverse mapping in order to correctly fill out the human-readable text form.

For messages to the participants, the challenge is to enable the manager to construct these textual and RDF/RDQL portions directly from the SEP declaration. Here there is a tension between the amount of RDF content that must be provided by the SEP author (in the template) vs. that provided by the SEP originator (when instantiating the template). Very specific SEP templates (e.g., to balance N people among appetizer, entree, and dessert choices) are the easiest to instantiate, because the author can specify the RDF terms needed in advance. General SEP templates (e.g., to balance N people among K arbitrary choices) are much more reusable, but require substantially more work to instantiate (and may require understanding RDF). Alternatively, authors may provide very general templates but make the specification of RDF terms for the choices optional; this enables easy template reuse but fails to provide semantic content for automated processing by the participants.

In our current system, we offer both highly specialized SEPs (e.g., for meeting scheduling) and more general SEPs (e.g., to give away some type of item). Enabling originators to easily customize general SEPs with precise semantic terms, perhaps from a set of offered ontologies, is an important area of future work.

6.4 Integrating with Non-Semantic Messages

Despite the advantages of semantic email, we do not want to create a strict dichotomy in our email habitat. In our potluck example, suppose that one of the participants wants to know whether there is organized transportation to the potluck (and this information affects his decision on what to bring). What should he do? Compose a separate non-semantic email to the originator and respond to the semantic one only later? A better (and easier to use) solution would be to treat both kinds of emails uniformly, and enable the participant to ask the question in replying to the semantic email, ultimately providing the *semantic* response later on in the thread.

Our implementation supports this behavior by supplying an additional *Remarks* field in each response form, where a participant may include a question or comment to be forwarded to the originator. For a question, the originator can reply, enabling the participant to respond to the original semantic question with the included form or pose another question.

6.5 Experience

Our semantic email system is deployed and may be freely used by anyone without any software installation; the source code for deploying other instances of the server is also available. So far we have developed simple processes for functions

like collecting RSVPs (i.e., confirming or declining an invitation), giving tickets away, scheduling meetings, and balancing a potluck. Our system uses standard ontologies where possible (e.g., RDF Calendar [56]), augmented as needed with a local semantic email schema.

Despite very limited publicity, our semantic email server has seen growing interest over the short time that it has been available. For instance, a DARPA working group has adopted semantic email for all of its meeting scheduling and RSVP needs, students have used semantic email to schedule seminars and Ph. D. exams, and semantic email has been used to organize our annual database group and departmental-wide potlucks. Furthermore, a number of other institutions have expressed interest in deploying copies of semantic email locally at their sites. These are merely anecdotes but lend credence to our claim that semantic email is both useful and practical.

Our semantic email system is integrated within our larger MANGROVE [38] semantic web system. This provides us with an RDF-based infrastructure for managing email data and integrating with web-based data sources and services. For instance, the MANGROVE web calendar accepts event information via email or from a web page. In addition, MANGROVE provides semantic email with an RDF data source about courses, people, etc. that could be used to support the prediction of likely responses by the manager discussed in Section 2. Likewise, a semantic email client could utilize data from MANGROVE to answer common questions. When previously unknown questions are answered manually by the user, these responses could be stored for future use, thus enabling the automatic acquisition of semantic knowledge over time. Future work will consider additional ways to synergistically leverage data from both the web and email worlds in MANGROVE.

7 Related Work

Information Lens [36] used forms to enable a user to generate a single email message with semi-structured content that might assist recipients with filtering and prioritizing that message. Our SEPs generalize this earlier work by enabling users to create an email *process* consisting of a set of interrelated messages, and by extending Information Lens's rule-based message processing to support more complex constraint and utility reasoning based on information from the entire set of messages. Consequently, SEPs support a much broader range of possible applications. More recently, Kalyanpur et al. [27] proposed having users semantically annotate messages to improve mail search, sorting, and filtering. This approach can potentially result in rich semantic content, but requires users to invest significant annotation effort for some *potential* future benefit (e.g., in improved searching for an old email) or primarily for the benefit of the recipient. SEPs instead generate both the semantic content and the text of the email message directly from simple forms, and provide instant gratification by immediately utilizing this content for simple

but time-saving email processes.

Our vision for semantic email was initially described in Etzioni et al. [17] and McDowell et al. [39]. Possible uses of semantic email are similar to those of some existing semantic web systems (e.g., [47,30,42], cf., RDF Calendar group discussions [56]). The key differentiating aspects of our work are its generality to many different tasks, its ability to interoperate freely with naive participants, and its polynomial time reasoning for recommending interventions. For instance, Rcal [47] uses messages between participants to agree upon meeting times and McIlraith et al. [42] describe an agent that makes travel arrangements by invoking various web services (which could be modeled as participants in a SEP). These systems, however, enable full interaction only between two parties that are both executing domain-specific software. For instance, though Rcal provides a web interface to let anyone schedule an appointment with an installed Rcal user, an Rcal user cannot use the system to request an appointment with a non-"Rcal-enabled" person. Likewise, McIlraith et al.'s agent is designed only to communicate with specific web services, not with humans (such as human travel agents) that could offer the same functionality. Our system instead permits processes to include any user, regardless of their capabilities. An additional, though less critical, distinction is our use of email instead of HTTP or a custom protocol (cf., Everyware [19]). Email provides a convenient transport mechanism because the vast majority of users already have well-known addresses (no additional directories are needed), messages can be sent regardless of whether the recipient has performed any configuration, and existing email clients provide a useful record of messages exchanged. Finally, our framework enables the automated pursuit of a wide variety of goals through reasoning in guaranteed polynomial time, a result not provided by the other systems discussed above. The combination of these factors makes semantic email a lightweight, general approach for automating many tasks that would be impractical with other systems.

Recent work on the *Inference Web* [41] has focused on the need to explain a Semantic Web system's *conclusions* in terms of base data and reasoning procedures. In contrast, we deal with explaining the SEP's *actions* in terms of existing responses and the expected impact on the constraints. In this sense our work is similar to prior research that sought to explain decision-theoretic advice (cf., Horvitz et al. [21]). For instance, Klein and Shortliffe [29] describe the VIRTUS system that can present users with an explanation for why one action is provided over another. Note that this work focuses on explaining the relative impact of multiple factors on the choice of some action, whereas we seek the simplest possible reason why some action could *not* be chosen (i.e., accepted). Other relevant work includes Druzdzel [16], which addresses the problem of translating uncertain reasoning into qualitative verbal explanations.

For constraint satisfaction problems (CSPs), a *nogood* [51] is a reason that no *current* variable assignment can satisfy all constraints. In contrast, our explanation

for a `PossiblyConstraint` is a reason that no *future* assignment can satisfy the constraints, given the set of possible future responses. Potentially, our problem could be reduced to nogood calculation, though a direct conversion would produce a problem that might take time that is exponential in N, the number of participants. However, for bounded constraints, we could create a CSP with variables based on the *aggregates* of the responses, rather than their specific values, as described in Section 3. Using this simpler CSP, we could then exploit existing, efficient nogood-based solvers (e.g., [25,28,24]) to find candidate explanations in time polynomial in N. Note though that most applications of nogoods have focused on their use for developing improved constraint solving algorithms [51,28] or for debugging constraint programs [46], rather than on creating explanations for average users. One exception is Jussien and Ouis [26], who describe how to generate user-friendly `nogood` explanations, though they require that a designer explicitly model a user's perception of the problem as nodes in some constraint hierarchy.

8 Conclusions

This paper generalizes the original vision of the semantic web to also encompass email. We have introduced a paradigm for semantic email and described a broad class of semantic email processes. These automated processes offer tangible productivity gains on email-mediated tasks that are currently performed manually in a tedious, time-consuming, and error-prone manner. Moreover, semantic email opens the way to scaling similar tasks to large numbers of people in a manner that is infeasible today. For example, large organizations could carry out surveys, auctions, and complex meeting coordination via semantic email with guarantees on the behavior of these processes.

Our technical contributions are as follows. We presented a formalization that teases out the issues involved, and used this formalization to explore several central inference questions. We then defined and explored two useful models for specifying the goals of a process and formalizing when and how the manager of the process should intervene. For our logical model we showed how the problem of deciding whether a response was ultimately acceptable relative to the constraints could be solved in polynomial time for bounded constraints. With our decision-theoretic model we addressed several shortcomings of the logical model and demonstrated how appropriate restrictions could enable the optimal policy for this model to be computed in polynomial time. In both cases we identified restrictions that greatly improved the tractability of the key reasoning problems while still enabling a large number of useful processes to be represented. In addition, we described how to automatically generate explanations for the manager's interventions and identified cases where these explanations can be computed in polynomial time. Finally, we described our publicly available semantic email system and how it satisfies the implementation desiderata of instant gratification, gradual adoption, and ease of use.

There are a number of interesting directions for future work. First, we want to consider interactions between semantic email and other semantic web applications to support more sophisticated reasoning techniques (e.g., check calendars and other resources to help constrain the number of messages and responses from some participants). We also plan to incorporate our recent work on schema and ontology mapping [14] to support more flexibility in responding to a semantic email message. In addition, our deployed system for semantic email offers the potential for a number of interesting user studies. For instance, it would be interesting to examine how originators make use of SEPs and which types are the most popular, the impact SEPs have on the efficiency of tasks compared to traditional email management, and how users react to interventions. Finally, we identified specific cases where our reasoning is tractable, but there are many opportunities for studying other cases within the framework we provided for modeling SEPs.

9 Acknowledgements

This research was supported in part by NSF grant IIS-0312988, DARPA contract NBCHD030010, and ONR grant N00014-02-1-0324 for Oren Etzioni; NSF CAREER grant IIS-9985114 and ITR grant IIS-0205635 for Alon Halevy; and by NSF Graduate Research and Microsoft Endowed Fellowships for Luke McDowell. Jeff Lin assisted with the server implementation. Thanks to Abraham Bernstein, Phil Bernstein, Sandy Liu, Mausam, Matthew Richardson, Stani Vlasseva, Dan Weld, and the anonymous reviewers for their helpful comments on improving this work.

A Proof Sketches for Logical SEPs

This section provides more details on the proofs for each of this paper's theorems related to L-SEPs. We assume throughout that an L-SEP Λ has N participants, a current state D, and constraints C_D, and that C_D refers to at most some constant H number of attributes.

A.1 Proof of Theorem 1

We first show that ultimate satisfiability is NP-complete in the general case. We then show how this problem can be solved in polynomial time when the constraints are either domain-bounded or constant-bounded.

NP-complete for arbitrary constraints: First, observe that ultimate satisfiability is in NP – given an L-SEP Λ and a response r, we can guess a possible outcome of D that is consistent with r, then verify that the outcome satisfies the constraints.

Second, we show that ultimate satisfiability is NP-hard via a reduction from 3-SAT. Assume we are given a boolean formula ϕ of the form $\phi = L_1 \wedge L_2 \wedge ... \wedge L_m$ where $L_i = (w_{i1} \vee w_{i2} \vee w_{i3})$ for $1 \leq i \leq m$, and each w_{ij} equals some variable x_k or $\overline{x_k}$ for $1 \leq k \leq n$. The 3-SAT problem is to determine if ϕ is satisfiable for some assignment to the variables of w.

Given ϕ, we construct an L-SEP Λ where:

- Participants $P = \{p_0, p_1, p_2, ..., p_n\}$
- Data set D is a single table with one attribute value
- Responses $R = \{nil, r_1, r_2, ...r_n\}$
- Constraints $C_D = \phi$, with v_{ij} substituted for each w_{ij} where
  if $w_{ij} = x_k$, then we set $v_{ij} = [(\text{COUNT(*) WHERE value = } r_k) > 0]$
  otherwise $w_{ij} = \overline{x_k}$, and $v_{ij} = [(\text{COUNT(*) WHERE value = } r_k) = 0]$

This construction is polynomial in the size of ϕ. In the resulting Λ, there are $n + 1$ participants that may each respond with one of $n + 1$ values.

Given this constructed Λ, we now show that the 3-SAT formula ϕ is satisfiable iff an initially empty D for Λ is ultimately satisfiable w.r.t. C_D given response nil. First, given an assignment $x_1, ..., x_n$ that satisfies ϕ, a final state of D that satisfies C_D is as follows: p_0 responds nil, p_k responds r_k if x_k is true, otherwise p_k responds nil. This will set the corresponding x_k's in C_D to true, and since ϕ is satisfied, C_D will be satisfied in the resultant state, demonstrating that D is ultimately satisfiable given an initial response nil. Alternatively, if D is ultimately satisfiable given initial response nil, we can take a final state of D that satisfies C_D and construct an assignment $x_1, ..., x_k$ that satisfies ϕ as follows: if any participant has responded with value r_k, then x_k is true; otherwise, x_k is false. Thus, any 3-SAT problem with N variables can be solved by reduction to ultimate satisfiability with $N + 1$ participants. Since 3-SAT is NP-complete in N, ultimate-satisfiability must be NP-hard in N.

Polynomial-time when constraints are *domain-bounded*: In this case, the constraints refer to attributes whose domain size is at most some constant L. Since there are most H attributes, there are thus a total of LH possible responses.

We evaluate the constraints over D', a data set that distinguishes only representative states that are different with respect to the constraints. In particular, all that matters for D' is the number of each type of response that has been received (i.e., aggregates of the responses). The number of possible states of D' is thus the number of ways of dividing N participants among $LH + 1$ possible responses (LH choices plus a "no response" option):

$$|D'| = \binom{N + LH}{LH} = O(N^{LH})$$

To determine ultimate satisfiability of D given r, we construct a data set D_r that is D augmented with the given response r. We then iterate over all possible values d

of D'. For each value, if d is inconsistent with D_r (i.e., for some response type R_i, D_r shows more such responses than d does), we discard d. Otherwise, we evaluate C_D over d – this requires time linear in N and $|C_D|$ given a particular d. Given this procedure, D is ultimately satisfiable for r iff some d is consistent with D_r and satisfies C_D. Each step requires linear time, and there are a polynomial number of iterations ($O(N^{LH})$), so the total time is polynomial in N and $|C_D|$.

Polynomial-time when constraints are *constant-bounded*: This case uses a similar algorithm as when the constraints are *domain-bounded*. However, since each attribute may have a potentially infinite domain, we must keep track of the possible states differently. Here, we allow only COUNT aggregations, which may be of the form: COUNT(*) WHERE value $= v_i$ or an inequality like COUNT(*) WHERE value $> v_i$.

If C_D is constant-bounded, then there are most K constants $v_1, ..., v_k$ used in these aggregations. These constants divide the domain of each attribute into at most $K+1$ regions. Thus, there are $K+1$ possibilities for each of the H attributes of a response, yielding a total of $O(KH)$ possible responses. As with the analysis above, the number of possible states in the representative data set D' is thus $O(N^{KH})$, and the time to evaluate each state is linear. Since H and K are assumed to be constants, then the total time to check ultimate satisfiability is polynomial in both N and $|C_D|$.

A.2 Proof of Theorem 2

For this theorem we are given an L-SEP Λ, current state D, and some `Possibly-Constraints` C_D, and wish to compute the acceptable set A of Λ. We consider the two cases where C_D is and is not bounded:

Polynomial time for bounded constraints: We can determine whether any particular response r is in A via testing ultimate satisfiability: r is in A iff D is ultimately satisfiable w.r.t. C_D for r. Since C_D is bounded, Theorem 1 states that this satisfiability testing can be done in time polynomial in N and the $|C_D|$. In addition, since C_D is bounded, either there are only a small number of possible responses (if C_D is domain-bounded), or there are only a bounded number of responses that are distinguishable w.r.t. the constraints (if C_D is constant-bounded, as discussed in the proof of Theorem 1)). In either case, there are only a constant number of different responses r that must be tested. Thus, by testing each representative response, we can determine the entire acceptable set (representing it as ranges of acceptable values) in time polynomial in N and $|C_D|$. If we actually construct the entire set A (as described in the theorem), then there is an additional polynomial time dependence on $|A|$.

NP-hard for arbitrary constraints: For this case we show that computing the acceptable set is NP-hard by a reduction from ultimate satisfiability: given an L-SEP Λ with N participants, data set D, constraints C_D, and a possible response r, Λ is ultimately satisfiable for r iff r is in the acceptable set A for Λ. This relationship follows directly from the definition of the acceptable set, and the reduction is clearly polynomial time. Since ultimate satisfiability is NP-complete in N for arbitrary constraints, computing the acceptable set must be NP-hard in N.

A.3 Proof of Theorem 3

Here we are given an L-SEP Λ, current state D, constraints C_D, and a response r, and wish to compute the minimum sufficient explanation E for rejecting r. This theorem has different results depending on whether C_D consists of `MustConstraints` or `PossiblyConstraints`:

Polynomial time for `MustConstraints`: For a `MustConstraint`, the size of the minimum sufficient explanation is always one. We can compute this explanation by adding r to D and then testing each constraint to see if it is unsatisfied in this new state; any such constraint is a minimum explanation. Testing each constraint on a given state is polynomial in N, and there are at most O($|C_D|$) constraints, for total time polynomial in N and $|C_D|$.

NP-hard for `PossiblyConstraints`: In this case computing a minimum explanation is NP-hard in two different ways. First, a reduction from ultimate satisfiability: given an L-SEP Λ, D, C_D, and r, D is ultimately satisfiable for r iff the minimum explanation for rejecting r on D does *not* exist. This relationship follows from the definition of an explanation, since if an explanation exists it rules out any way of satisfying the constraints, and the reduction is clearly polynomial. Thus, since determining ultimately satisfiability is NP-complete in N (Theorem 1), then computing the minimum explanation is NP-hard in N.

Second, a reduction from SET-COVER, which is defined as follows: We are given a set $X = \{1, 2, ..., N\}$ and family of subsets of $F = \{S_1, S_2, ..., S_M\}$ such that every $S_i \subset X$ and every element of X is contained in some S_i. A cover for this problem is a set $F' \subset F$ such that the union of all $S_i \in F'$ contains every element of X. The problem is to determine whether there exists a cover of size J or smaller for X.

We construct the following L-SEP Λ with:

- **Participants:** $P = \{p_0, p_1, p_2, ..., p_N\}$.
- **Data set:** D is a single table with one boolean attribute R.
- **Constraints:** a set of `PossiblyConstraints` $C_D = C_0 \wedge C_1 \wedge C_2 \wedge ... \wedge C_M$

where

$$C_0 = (R_{yes} = 0)$$
$$C_i = \bigwedge_{j \in S_i} (R_{true} \neq j) \; for \; 1 \leq i \leq M$$
$$R_{yes} = (COUNT \; (*) \; WHERE \; \mathsf{value} = True)$$

Constructing this L-SEP is clearly polynomial time in the size of the SET-COVER problem.

Given this construction for Λ, we now show that a set cover for X of size J exists iff the minimum explanation E for rejecting a response r of `False` for Λ with an initially empty state D contains $J + 1$ constraints. First, given an explanation E with $J + 1$ constraints, a minimum cover F' is the set of all S_i such that C_i is present in E, for $i \neq 0$. (Every sufficient explanation E contains C_0; it is a special case included just to handle the situation where all participants respond `No`. Hence, F' will be of size J.) To see why this works, consider an example set $S_7 = \{3, 5\}$. This set is mapped to the constraint $C_7 = (R_{true} \neq 3) \wedge (R_{true} \neq 5)$. A sufficient explanation for rejecting r must cover every possible outcome of the L-SEP, and two such outcomes are for either 3 or 5 participants to respond `True`. Thus, if response r is to be rejected, the explanation E must cover these two cases, either by choosing C_7, or by choosing some other constraint(s) that also covers the cases of 3 or 5 `True` responses. This follows exactly the same rules as a solution to SET-COVER. Likewise, given a cover F' for X of size J, a minimum explanation for rejecting an initial `False` response is the conjunction of C_0 together with all constraints C_i where S_i is in F', for a total size of $J+1$. Thus, any input to the SET-COVER problem can be reduced to solving the minimum explanation problem. Since the former problem is NP-complete in the number of sets (M), the latter problem must also be NP-hard in number of constraints ($|C_D|$). Combining this with the previous result, we see that computing the minimum sufficient explanation for `PossiblyConstraints` is NP-hard in N and NP-hard in $|C_D|$.

A.4 Proof of Theorem 4

We are given an L-SEP Λ with N participants, current state D, constraints C_D, and a response r and wish to find the minimum sufficient explanation E for rejecting r, assuming that C_D is bounded and that the size of a minimum E is no more than some constant J. If C_D consists of `MustConstraints`, then we already know that this problem is polynomial time in N and $|C_D|$ from Theorem 3.

If C_D is made up of `PossiblyConstraints`, then we can test if any particular explanation E is a sufficient explanation via ultimate satisfiability: E is a sufficient explanation iff $E \subseteq C_D$ and D is *not* ultimately satisfiable w.r.t. E for r. Since the constraints are bounded, Theorem 1 states that this testing can be performed in time

polynomial in N and $|C_D|$. In addition, since any minimum explanation E contains only terms from C_D, restricting E to at most size J means that the total number of explanations that must be considered is polynomial in $|C_D|$. Thus, we can compute the minimal explanation by testing the sufficiency of every possible explanation of size J or less and picking the smallest sufficient explanation. This algorithm runs in total time polynomial in N and $|C_D|$.

B Proof Sketches for Decision-theoretic SEPs

This section provides proofs regarding the complexity of computing the optimal policy for a given D-SEP δ with N participants. We assume that each participant will eventually send an original response, then only sends further messages if they receive a suggestion (which they will also eventually respond to). For convenience, we define the following notation: OPTPOLICY(δ) is the problem of determining the optimal policy $\pi^\star$ for a given D-SEP δ. OPTUTILITY(δ,θ) is the problem of determining if the expected total utility of $\pi^\star$ for δ exceeds some constant θ.

B.1 Proof of Theorem 5 – bounded suggestions

For this first case, we assume that the manager can send at most some constant L messages to each participant. Below we prove that in this case OPTUTILITY(δ,θ) is PSPACE-complete, then use this result to prove that OPTPOLICY(δ) is PSPACE-hard.

OPTUTILITY(δ,θ) is PSPACE-complete: First, we show that OPTUTILITY(δ,θ) is in PSPACE. Given δ, consider the tree representing all possible executions, where the root of the tree is the initial state and each leaf represents a possible halted state. From any state in the tree, the next state may result either from the manager making a suggestion or from receiving a response from some participant. Hence, the branching factor of the tree is $O(N)$. In addition, since the manager may make at most LN suggestions and each participant may send up to $L + 1$ responses, the tree is acyclic and has total height $O(LN)$. Consequently, we can determine the expected utility of the optimal policy via a suitable depth-first search of the tree. Since the utility of a child node can be discarded once the expected utility of its parent is known, the total space needed is just $O(LN)$. Thus, OPTUTILITY(δ,θ) is in PSPACE.

Second, we show that OPTUTILITY(δ,θ) is PSPACE-hard by a reduction from QBF (quantified boolean formula). A QBF problem specifies a formula φ of the form:

$$\varphi = \exists x_1 \forall y_1 ... \exists x_k \forall y_k \; \phi$$

where ϕ is a 3-CNF boolean formula over the x_i's and y_i's. The computational problem is to determine if φ is true.

Given φ, we construct a corresponding D-SEP δ as follows:

- **Participants:** $P = \{A_1, ..., A_k, B_1, ..., B_k\}$, for a total of $N = 2k$ participants.
- **States:** A state $s = (a_1, ..., a_k, b_1, ..., b_k)$ where the a_i's and b_i's indicate each participant's current response ($True$, $False$, or $NoneYet$). The a_i's and b_i's correspond directly to the x_i's and y_i's in the formula ϕ. Thus, we say "ϕ is satisfied in s" if no a_i or b_i has the value $NoneYet$ and evaluating ϕ by substituting corresponding values for the x_i's and y_i's yields true.
- **Values:** $V = \{True, False\}$.
- **Actions:** $A = \{NoOp, Halt, SW_{p,true}, SW_{p,false}\}$, where $p \in P$.
- **Transitions:** We construct $T()$ so that the following steps will occur in order:
  (1) **Choice:** In the initial state the manager may either perform $NoOp$ (to wait for responses) or $Halt$ (if it has no winning strategy).
  (2) **A-Turn:** A_1 sends a $False$ response. The manager may choose either to execute $NoOp$ (thus accepting $a_1 = False$) or to suggest a change to A_1, in which case A_1 immediately agrees (so $a_1 = True$).
  (3) **B-Turn:** The manager performs $NoOp$, and receives an random original response (either $True$ or $False$) from B_1. B_1 refuses any suggestions.
  (4) **Repeat:** Repeat A-Turn and B-Turn for (A_2, B_2) ... (A_k, B_k), then $Halt$.
- **Utilities:** the only non-zero utilities are as follows:

$$U(s_0, Halt) = 1 \;\; (quitting\ from\ the\ initial\ state)$$
$$U(s, Halt) = 1 + \epsilon \;\; if\ s \neq s_0 \;\; and\ \phi(s) = True$$

  where ϵ represents an infinitesimally small, positive value. Note that this use of ϵ does not introduce any serious computational difficulties. The expected utility of each state may be maintained in the form $(c + d\epsilon)$ – addition, multiplication, and comparison (over a total order) are easily defined for such values. In addition, since ϵ appears only in the utility function, higher-order values such as ϵ^2 do not arise.

The size of this D-SEP is polynomial in N and the whole reduction can be done in polynomial time. In particular, while an explicit representation of the transition and utility functions for every possible state would be exponential in N, the rules above allow all of the necessary functionality to be encoded concisely in terms of the current responses. For instance, the utility function representing one possibility for a B-turn (where b_i changes from $NoneYet$ to $True$) is:

$$T(\ s\ , NoOp, s') = 0.5\ where$$
$$s = \{a_1, ..., a_i, None_{i+1}, ..., None_k, b_1, ..., b_{i-1}, \mathbf{None_i}, None_{i+1}..., None_k\}$$
$$s' = \{a_1, ..., a_i, None_{i+1}, ..., None_k, b_1, ..., b_{i-1}, \mathbf{True}, None_{i+1}, ..., None_k\}$$

Note also that in several steps above we made statements like "The manager performs action $NoOp$," when really at each step the manager has a choice to make. However, since we can construct the transition function in any desired fashion, we can "force" the manager into any needed behavior by setting the transition probability for executing any other action to zero. The same control over the probabilities

permits us to ensure that participants behave in certain ways and that messages arrive in a certain order.

We now demonstrate an additional result needed to complete the proof:

Definition B.1 (**guaranteed satisfying policy**) Given a D-SEP δ constructed from φ as above, a guaranteed satisfying policy is a policy that, if followed by the manager, guarantees that the SEP will terminate in a state that satisfies ϕ. □

Claim: A guaranteed satisfying policy for δ exists iff the expected utility of the optimal policy $\pi^\star$ for δ is greater than 1 (e.g., OPTUTILITY(δ,$\theta = 1$) is true).

Proof: Clearly, the expected utility of a guaranteed satisfying policy for δ is $1 + \epsilon$, so any optimal policy must have utility at least this large, which is greater than 1. In the other direction, by examining the utility function we see that the only way for $\pi^\star$ to obtain a utility greater than 1 is for the SEP to halt with ϕ satisfied, yielding reward $1 + \epsilon$. If this outcome occurs with any probability $P_\phi < 1$ for $\pi^\star$, then the total expected utility will be less than 1. Thus, if the expected utility of $\pi^\star$ is greater than 1, some guaranteed satisfying policy must exist. □

Finally, we show that the QBF formula φ is true iff a guaranteed satisfying policy for δ exists. In the D-SEP, the manager can choose whether to set each a_i true or false by making a suggestion or not when A_i sends its response. This corresponds to the "exists" quantifications in φ – when trying to prove the formula true, we can choose any desired value for x_i. On the other hand, the manager cannot influence the values of b_i – these are chosen at random. Thus, the manager will have a guaranteed satisfying policy iff it's policy can handle all possible choices of the b_i's. This corresponds exactly to the "for all" quantifications of the y_i's. Note that we don't depend on the precise values of the probabilities – all that matters is that both true and false can occur for each b_i with some positive probability. Thus, a guaranteed satisfying policy for δ exists iff the QBF formula is true. Since the latter problem is PSPACE-complete, then the problem of determining if δ has a guaranteed satisfying policy is PSPACE-hard, and hence (by the above claim) OPTUTILITY(δ,θ) for a bounded number of suggestions must also be PSPACE-hard.

OPTPOLICY(δ) is PSPACE-hard: We show that OPTPOLICY(δ) is PSPACE-hard by reducing from OPTUTILITY(δ,θ). Given a D-SEP δ, we construct δ' to be the same as δ except that it has a new initial state s'_0. From s'_0, the manager may choose $Halt$ in order to end the process and gain utility $\theta + \epsilon$, or may choose $NoOp$, in which case the process transitions to the original initial state s_0. This construction is easy to do and runs in polynomial time. The original D-SEP δ has an expected utility for $\pi^\star$ that exceeds θ iff the optimal policy for δ' specifies that the manager should perform the initial action $NoOp$. This follows since if the expected utility of δ is θ or less, the optimal decision is to $Halt$ immediately, taking the utility $\theta + \epsilon$. Thus, since OPTUTILITY(δ,θ) is PSPACE-complete for a bounded number

of suggestions, the corresponding problem of OPTPOLICY(δ) must be PSPACE-hard.

B.2 Proof of Theorem 5 – unlimited suggestions

For this second case, we assume that the manager may make an unlimited number of suggestions to any participant. Below we prove that in this case OPTUTILITY(δ,θ) is EXPTIME-complete, then use this result to prove that OPTPOLICY(δ) is EXPTIME-hard.

OPTUTILITY(δ,θ) is EXPTIME-complete : First, we show that OPTUTILITY(δ,θ) is in EXPTIME. Given a D-SEP δ, we can convert δ into a Markov Decision Process (MDP) with $O(N)$ possible actions and one state for each state in δ. The MDP can be then solved with techniques such as linear programming that run in time polynomial in the number of states [35]. For δ, the number of states is exponential in N, so the total time is exponential. Then the expected utility of $\pi^\star$ for δ exceeds θ iff the optimal value of the initial state of the MDP exceeds θ.

Second, we show that OPTUTILITY(δ,θ) is EXPTIME-hard by a reduction from the game G_4 [53]. This game operates as follows (description from [34]): The "board" is a 13-DNF (disjunctive normal form) formula φ with a set of assignments to its $2k$ boolean variables. One set of variables $x_1, ..., x_k$ belong to player 1 and the rest $y_1, ..., y_k$ to player 2. Players take turns flipping the assignment of one of their variables. The game is over when the 13-DNF formula evaluates to true with the winner being the player whose move caused this to happen. The computational problem is to determine whether there is a winning strategy for player 1 for a given formula from a given initial assignment of the variables. Without loss of generality, below we assume that the original formula has been transformed so that the corresponding initial assignment sets all variables to false.

Given an instance of the game G_4 over some 13-DNF formula φ, we construct a corresponding D-SEP δ as follows:

- **Participants:** $P = \{A_1, ..., A_k, B_1, ..., B_k\}$, for a total of $N = 2k$ participants.
- **States:** A state $s = (a_1, ..., a_k, b_1, ..., b_k, Pend, Last)$ where the a_i's and b_i's indicate each participant's current response ($True, False$, or $NoneYet$), $Pend$ is the set of participants that the manager has made a suggestion to that has not been responded to yet, and $Last$ indicates whether the last message that changed a value was from some A or some B. The a_i's and b_i's correspond directly to the x_i's and y_i's in the formula φ. Thus, we say "φ is satisfied in s" if no a_i or b_i has the value $NoneYet$ and evaluating φ by substituting corresponding values for the x_i's and y_i's yields true.
- **Values:** $V = \{True, False\}$.

- **Actions:** $A = \{NoOp, Halt, SW_{p,true}, SW_{p,false}\}$ where $p \in P$.
- **Transitions:** We construct $T()$ so that the following steps will occur in order:
  (1) **Choice:** In the initial state the manager may either perform $NoOp$ (to wait for responses) or $Halt$ (if it has no winning strategy).
  (2) **Startup:** Every participant sends in a response $False$. The manager then suggests a change to every B_i, who do not immediately respond.
  (3) **A-Turn:** The manager chooses some A_i to suggest a change to. A_i immediately agrees, flipping the current value of a_i. If φ is now satisfied, $Halt$.
  (4) **B-Turn:** The manager performs $NoOp$, and receives a response to a previous suggestion from some random B_i, flipping the value of b_i. The manager immediately sends another suggestion back to the same B_i, who does not yet respond. If φ is now satisfied, $Halt$. Otherwise, go back to A-Turn.
- **Utilities:** the only non-zero utilities are as follows:
  $$U(s_0, Halt) = 1 \;\; (quitting\ from\ the\ initial\ state)$$
  $$U(s, Halt) = 1 + \epsilon \;\; if\ s \neq s_0,\ s.Last = A,\ and\ \varphi(s) = True$$

The size of this D-SEP is polynomial in N and the whole reduction can be done in polynomial time. As with the bounded suggestions case, the explicit transition and utility functions are exponential in N, but the rules above allow all of the necessary cases to be represented concisely in terms of the current responses, $Pend$, and $Last$. Likewise, we can "force" the needed manager and participant behavior by appropriate setting of the transition function.

We now demonstrate an additional result needed to complete the proof:

Definition B.2 (**guaranteed A-Win policy**) Given a D-SEP δ constructed from φ as above, a guaranteed A-Win policy is a policy that, if followed by the manager, guarantees that the SEP will terminate in a state that satisfies φ and where the last step was an "A-Turn." □

Claim: A guaranteed A-Win policy for δ exists iff the expected utility of the optimal policy $\pi^\star$ for δ is greater than 1 (e.g., OPTUTILITY(δ,$\theta = 1$) is true).

Proof: Analogous to the claim previously given for a guaranteed satisfying policy in the bounded suggestions case.□

Finally, we show that a winning strategy exists for player 1 in G_4 iff a guaranteed A-Win policy exist for δ. We consider the possible actions for the SEP manager, who represents player 1. In the initial "Choice" step, if the manager does not have a guaranteed A-Win policy, it is best to $Halt$ immediately and settle for a utility of 1. If the manager decides to play, then it also has a choice in Step 3 of which A_i to suggest a change to – this corresponds to choosing which x_i for Player 1 to flip. Step 4 corresponds to Player 2's flip of some y_i, and the manager has no choice to make. Thus, given a winning strategy for Player 1 in G_4, it is easy to construct

a guaranteed A-Win policy for δ (mapping x_i flips to A_i change suggestions), and vice versa. Since the problem of determining if Player 1 has such a winning strategy for G_4 is EXPTIME-hard, the problem of determining if δ has a guaranteed A-Win policy is EXPTIME-hard, and hence (by the above claim) the problem of OPTUTILITY(δ,θ) must also be EXPTIME-hard.

OPTPOLICY(δ)is EXPTIME-hard: This proof follows exactly the same form as the proof of OPTPOLICY(δ) for the bounded suggestions case. Since OPTUTILITY(δ,θ) is EXPTIME-complete for an unlimited number of suggestions, the corresponding problem of OPTPOLICY(δ) must be EXPTIME-hard.

B.3 Proof of Theorem 6

Here we show how to compute the optimal policy $\pi^\star$ in time polynomial in N, assuming a K-partitionable utility function and that the manager sends at most one suggestion to any participant. Although the formalisms are very different, the key observation underlying this proof is similar to that of Theorem 1. Here we also create a state space that only models the *number* of participants in each group, rather than their specific members.

We define a summary state function $S = \{\bar{C}, \bar{D}, \bar{E}\}$ where

- $\bar{C} = (C_1, ..., C_K)$ where C_i is the number of responses V_i that were received that do *not* have a suggestion pending.
- $\bar{D} = (D_1, ..., D_K)$ where D_i is the number of responses V_i that were received that *do* have a suggestion pending.
- $\bar{E} = (E_1, ..., E_K)$ where E_i is the number of responses V_i that were received as a response to a suggestion.

In what follows, the notation $\bar{C} - v$ indicates "subtract one from the variable in $\bar{C}$ specified by value v." Given S, we can define the following transitions (omitting details for states where everyone has already responded):

$$T(\{\bar{C}, \bar{D}, \bar{E}\}, SW_v, \ \{\bar{C}-v, \bar{D}+v, \bar{E} \quad \}) = \quad 1$$

$$T(\{\bar{C}, \bar{D}, \bar{E}\}, NoOp, \{\bar{C}+v, \bar{D}, \quad \bar{E} \quad \}) = \ \rho_o(\bar{C}, \bar{D}, \bar{E}) \cdot \rho_v$$

$$T(\{\bar{C}, \bar{D}, \bar{E}\}, NoOp, \{\bar{C}, \quad \bar{D}-v, \bar{E}+w\}) = \rho_{sv}(\bar{C}, \bar{D}, \bar{E}) \cdot \rho_{vw}$$

The first equation represents the manager requesting that some respondent switch their response from the value v; the state is updated to note that a suggestion has been made (with probability 1). The next two equations handle the uncertainty

when the manager decides to wait for the next message to arrive. Specifically, the second equation handles the case when the next message is an original response from a previously unheard from participant (probability $\rho_o(\bar{C}, \bar{D}, \bar{E})$), while the third equation handles the case where the next message is a response to a previously made suggestion to switch from value v (probability $\rho_{sv}(\bar{C}, \bar{D}, \bar{E})$).

At any time each participant's response is either counted once among the K variables of each of $\bar{C}$, $\bar{D}$, or $\bar{E}$, or has not yet been received. The number of possible states is thus the number of ways of dividing N participants among $3K + 1$ groups, which is:

$$|S| = \binom{N + 3K}{3K} = O(N^{3K})$$

Because of the restriction to send at most one suggestion to each participant, the graph formed by the transition function over these states is acyclic. Thus, the optimal policy may be computed via a depth-first search over the graph in total time $O(N^{3K})$.

References

[1] S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *PODS*, 1998.

[2] A. Ankolenkar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. DAML-S: A semantic markup language for web services. In *Proceedings of the Semantic Web Working Symposium*, pages 411–430, 2001.

[3] F. Baader and U. Sattler. Description logics with concrete domains and aggregation. In *Proeedings of the European Conference on Artificial Intelligence*, pages 336–340, 1998.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[5] D. Bertsekas. Dynamic programming and optimal control. Athena Scientific, 1995.

[6] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Artificial Intelligence Planning Systems*, pages 52–61, 2000.

[7] C. Boutilier. A POMDP formulation of preference elicitation problems. In *AAAI-02*, pages 239–246, 2002.

[8] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proc. of IJCAI-14*, 1995.

[9] J. A. Boyan and M. L. Littman. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems 13 (NIPS)*, pages 1026–1032, 2000.

[10] G. Carenini and J. D. Moore. Generating explanations in context. In *Intelligent User Interfaces*, pages 175–182, 1993.

[11] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of PODS*, pages 155–166, 1999.

[12] M. G. de la Banda, P. J. Stuckey, and J. Wazny. Finding all minimal unsatisfiable subsets. In *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 32–43. ACM Press, 2003.

[13] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: a machine learning approach. In *Proc. of SIGMOD*, 2001.

[14] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *WWW*, 2002.

[15] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[16] M. Druzdzel. Qualitative verbal explanations in bayesian belief networks. *Artificial Intelligence and Simulation of Behaviour Quarterly*, 94:43–54, 1996.

[17] O. Etzioni, A. Halevy, H. Levy, and L. McDowell. Semantic email: Adding lightweight data manipulation capabilities to the email habitat. In *Sixth Int. Workshop on the Web and Databases*, 2003.

[18] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. M.I.T Press, 1995.

[19] F. Ferreira, D. Schwabe, and C. Lucena. Using semantic web services now. In *IX Brazilian Symp. on Hypermedia and Multimedia*, 2003.

[20] S. Grumbach and L. Tininini. On the content of materialzed aggregate views. In *Proc. of PODS*, 2000.

[21] E. J. Horvitz, J. S. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2:247–302, 1988.

[22] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A look behind the curtain. In *PODS*, 2003.

[23] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[24] U. Junker. QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, Seattle, WA, USA, Aug. 2001.

[25] N. Jussien and V. Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, Sept. 2000.

[26] N. Jussien and S. Ouis. User-friendly explanations for constraint programming. In *ICLP 11th Workshop on Logic Programming Environments*, Paphos, Cyprus, Dec. 2001.

[27] A. Kalyanpur, B. Parsia, J. Hendler, and J. Golbeck. SMORE - semantic markup, ontology, and RDF editor. http://www.mindswap.org/papers/.

[28] G. Katsirelos and F. Bacchus. Unrestricted nogood recording in csp search. In *Principles and Practice of Constraint Programming*, October 2003.

[29] D. A. Klein and E. H. Shortliffe. A framework for explaining decision-theoretic advice. *Artificial Intelligence*, 67(2):201–243, 1994.

[30] S. Kumar, A. Kunjithapatham, M. Sheshagiri, T. Finin, A. Joshi, Y. Peng, and R. S. Cost. A personal agent application for the semantic web. In *AAAI Fall Symposium on Personalized Agents*, 2002.

[31] N. Kushmerick, R. Doorenbos, and D. Weld. Wrapper induction for information extraction. In *Proc. of IJCAI*, 1997.

[32] O. Lassila and R. Swick. Resource description framework (RDF) model and syntax specification. http://www.w3.org/TR/REC-rdf-syntax/, 1999. W3C Recommendation.

[33] A. Y. Levy, I. S. Mumick, and Y. Sagiv. Query optimization by predicate move-around. In *Proc. of VLDB*, pages 96–107, 1994.

[34] M. L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proce. of (AAAI-97)*, 1997.

[35] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving Markov decision problems. In *Proc. of (UAI–95)*, 1995.

[36] T. Malone, K. Grant, F. Turbak, S. Brobst, and M. Cohen. Intelligent information-sharing systems. *Comm. of the ACM*, 30(5):390–402, 1987.

[37] B. McBride. Jena: Implementing the RDF model and syntax specification. In *Proceedings of the 2001 Semantic Web Workshop*, 2001.

[38] L. McDowell, O. Etzioni, S. D. Gribble, A. Halevy, H. Levy, W. Pentney, D. Verma, and S. Vlasseva. Mangrove: Enticing ordinary people onto the semantic web via instant gratification. In *Second International Semantic Web Conference*, October 2003.

[39] L. McDowell, O. Etzioni, A. Halevey, and H. Levy. Semantic email. In *Proc. of the Thirteenth Int. WWW Conference*, 2004.

[40] D. L. McGuinness and A. Borgida. Explaining subsumption in description logics. In *IJCAI (1)*, pages 816–821, 1995.

[41] D. L. McGuinness and P. Pinheiro da Silva. Infrastructure for web explanations. In *Second International Semantic Web Conference*, October 2003.

[42] S. A. McIlraith, T. C. Son, and H. Zeng. Mobilizing the semantic web with daml-enabled web services. In *Proceedings of the 2001 Semantic Web Workshop*, 2001.

[43] C. Mohan. Workflow management in the internet age. www.almaden.ibm.com/u/mohan/workflow.pdf, 1999.

[44] S. Mukherjee, H. Davulcu, M. Kifer, P. Senkul, and G. Yang. Logic based approaches to workflow modeling and verification. In *Logics for Emerging Applications of Databases*, 2003.

[45] R. Neches, W. R. Swartout, and J. D. Moore. Explainable (and maintainable) expert systems. In *IJCAI*, pages 382–389, 1985.

[46] S. Ouis, N. Jussien, and P. Boizumault. k-relevant explanations for constraint programming. In *FLAIRS'03: Sixteenth International Florida Artificial Intelligence Research Society Conference*, St. Augustine, Florida, USA, May 2003. AAAI press.

[47] T. Payne, R. Singh, and K. Sycara. Calendar agents on the semantic web. *IEEE Intelligent Systems*, 17(3):84–86, 2002.

[48] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th Annual IEEE Symposium on Foundations of Computer Science*, 1977.

[49] M. Puterman. Markov decision processes. Wiley Inter-science, 1994.

[50] K. Ross, D. Srivastava, P. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. In *Principles and Practice of Constraint Programming*. LNCS, 874. Springer Verlag, 1994.

[51] T. Schiex and G. Verfaillie. Nogood Recording fot Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, 3(2):187–207, 1994.

[52] P. Senkul, M. Kifer, and I. H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *VLDB*, 2002.

[53] L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.

[54] W. Swartout, C. Paris, and J. Moore. Design for explainable expert systems. *IEEE Expert*, 6(3):58–647, 1991.

[55] V. Tamma, M. Wooldridge, and I. Dickinson. An ontology-based approach to automated negotiation. In *AMEC-2002*, 2002.

[56] W3C. RDF calendar workspace. http://www.w3.org/2002/12/cal/.

[57] W3C. Web-ontology (webont) working group. http://www.w3.org/2001/sw/WebOnt/.