University of Dayton
eCommons

Computer Science Faculty Publications

Department of Computer Science

10-2016

Mining Mixed-Initiative Dialogs

Saverio Perugini
University of Dayton, sperugini1@udayton.edu

Follow this and additional works at: http://ecommons.udayton.edu/cps_fac_pub

Part of the Other Computer Sciences Commons

eCommons Citation

Perugini, Saverio, "Mining Mixed-Initiative Dialogs" (2016). *Computer Science Faculty Publications.* Paper 59.
http://ecommons.udayton.edu/cps_fac_pub/59

This Conference Paper is brought to you for free and open access by the Department of Computer Science at eCommons. It has been accepted for inclusion in Computer Science Faculty Publications by an authorized administrator of eCommons. For more information, please contact frice1@udayton.edu, mschlangen1@udayton.edu.

Mining Mixed-initiative Dialogs

Saverio Perugini
Department of Computer Science
University of Dayton
300 College Park
Dayton, Ohio 45469–2160 USA
Tel: +001 (937) 229–4079 Fax: +001 (937) 229–2193
E-mail: saverio@udayton.edu
WWW: http://academic.udayton.edu/SaverioPerugini

***Abstract*—Human-computer dialogs are an important vehicle through which to produce a rich and compelling form of human-computer interaction. We view the specification of a human-computer dialog as a set of sequences of progressive interactions between a user and a computer system, and mine partially ordered sets, which correspond to mixing dialog initiative, embedded in these sets of sequences—a process we refer to as *dialog mining*—because partially ordered sets can be advantageously exploited to reduce the control complexity of a dialog implementation. Our mining losslessly compresses the specification of a dialog. We describe our mining algorithm and report the results of a simulation-oriented evaluation. Our algorithm is sound, and our results indicate that it can compress nearly all dialog specifications, and some to a high degree. This work is part of broader research on the specification and implementation of mixed-initiative dialogs.**

I. INTRODUCTION

From application software to the web, interactive computing systems involve progressive interactions between the user and the system in support of the completion of a particular task or satisfaction of a specific goal. These progressive interactions are considered *dialogs* between a human and a computer system. Viewed from this perspective, posting an item for sale on *eBay*, completing a transaction at an ATM, purchasing a product from *Amazon*, and plotting a chart or graph in Microsoft Excel are task scenarios involving human-computer dialog. Dialogs are deemed an important vehicle through which to produce a rich and compelling form of human-computer interaction.

Dialogs vary widely from those that keep the user in lock-step with the initiative of the system to those which impart some to equal control to the user over the direction in which to steer the dialog, called *mixed-initiative dialogs* [1], [2], [3]. In this article, we view the specification of a human-computer dialog as a set of sequences of interaction steps from the start of a dialog through completion. We are concerned with mining (i.e., identifying) partially ordered sets, which correspond to mixing dialog initiative, embedded in these sets of sequences because partially ordered sets can be advantageously exploited to reduce the control complexity of the implementation of a dialog. The left side of Fig. 1 provides a conceptual overview of our approach. We start with a high-level specification of a human-computer dialog and mine it for a compressed representation which captures the requirements of the dialog—a process we refer to as *dialog mining*—in preparation for implementation of the dialog (see right side of Fig. 1).

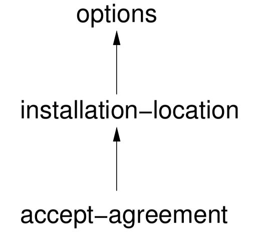
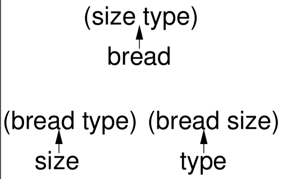
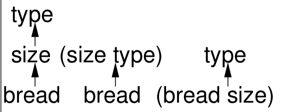
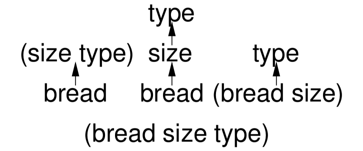
II. HUMAN-COMPUTER DIALOGS

A. Fixed- and Mixed-initiative Dialogs

Consider a dialog in a wizard for installing application software [4]. Typically the user must first accept the license agreement, then select an installation location (i.e., disk and directory), and finally choose options (e.g., which components of the software to install). Such a dialog is a *fixed* dialog due to the fixed order of the questions from which the user is not permitted to deviate in his responses [5]. An *enumerated specification* is a set of episodes, and an *episode* is an ordered list of questions to be posed and answered from the start of the dialog to completion. Intuitively, an enumerated specification is a set of all possible ways to complete a dialog. An enumerated specification of this installation wizard dialog is $\{\prec\text{accept-agreement installation-location options}\succ\}$. Intuitively, an enumerated specification constitutes a plan for a dialog. Formally, a dialog specification is a set of *totally ordered sets*. We use a *Hasse* diagram, a graphical depiction of a *partially ordered set*, to represent a dialog specification. A relation R with the set S over whose Cartesian product R is defined is a *strict partially ordered set* (or *poset*) if R is an irreflexive, asymmetric, and transitive relation. This means that some of the elements of S may be unordered based on R. For instance, the set $\{(a,c),(b,d)\}$ is a strict partially ordered set. On the other hand, a relation R with the set S over whose Cartesian product R is defined is a *strict totally ordered set* if and only if for every two elements $(x,y) \in S$, xRy or yRx. For instance, the set $\{(a,b),(b,c),(a,c)\}$ is a strict totally ordered set. Every totally ordered set is also a partially ordered set, but the reverse is not necessarily true. Column (a) of Table Ia illustrates the Hasse diagram that specifies this software installation dialog. A Hasse diagram is read bottom-up. Here, the set S of the poset is the set of the questions posed in the dialog and R is the 'must be answered before' relation denoted with an upward arrow between the source and target of the arrow.

Flexible dialogs typically support multiple completion paths. For example, consider a dialog for ordering at the sandwich shop *Subway*. The customer must select a bread

←— (most rigid) fixed dialogs ……………… **dialogs between fixed dialogs and complete, mixed-initiative dialogs** ………………

ID	**(a)**	**(b)**	**(c)**	**(d)**	**(e)**
Enumerated Specification	{≺accept-agreement installation-location options≻}	{≺(bread size type)≻}	{≺bread (size type)≻, ≺size (bread type)≻, ≺type (bread size)≻}	{≺(bread size type)≻, ≺(bread (size type)≻, ≺(bread size) type≻}	{≺(bread size type)≻, ≺bread (size type)≻, ≺(bread size) type≻, ≺bread size type≻}
Hasse diagram(s)	 options installation–location accept–agreement	(bread size type)	 (size type) bread (bread type) (bread size) size type	 type size (size type) type bread bread (bread size)	 type (size type) size type bread bread (bread size) (bread size type)

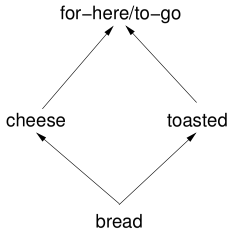
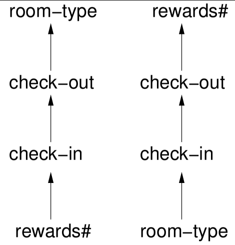
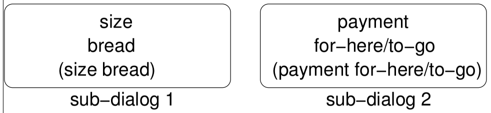
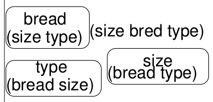
Output	`((C accept–agreement installation–location options))`	`((I bread size type))`	`((C bread (I size type))`
			`(C size (I bread type))`
			`(C type (I bread size)))`
				`((C bread (PFAN ∗ size type))`
				`(C (I bread size) type))`
					`(PFAN ∗ bread size type))`

Ratio	(1:1) 1:1	(1:1) 1:1	(3:3) $q{:}q$=1:1	(3:2) q:?	$(2^{3-1=2} = 4{:}1)\ 2^{q-1}{:}1$

(a) Beginning of a space of dialogs, continued in table (b) below.

……………… **dialogs between fixed dialogs and complete, mixed-initiative dialogs** ………………

complete,
mixed-initiative dialogs (most flexible) ⟶

ID	**(f)**	**(g)**	**(h)**	**(i)**	**(j)**	**(k)**
Enumerated Specification	{≺bread cheese toasted for-here/to-go≻, ≺bread toasted cheese for-here/to-go≻}	{≺rewards# check-in check-out room-type≻, ≺room-type check-in check-out rewards#≻}	{≺size bread payment for-here/to-go≻, ≺size bread for-here/to-go payment≻, ≺(size bread) for-here/to-go payment≻, ≺(size bread) payment for-here/to-go≻, ≺bread size payment for-here/to-go≻, ≺bread size for-here/to-go payment≻, ≺payment for-here/to-go size bread≻, ≺payment for-here/to-go bread size≻, ≺for-here/to-go payment size bread≻, ≺for-here/to-go payment bread size≻, ≺bread size (payment for-here/to-go)≻, ≺size bread (payment for-here/to-go)≻, ≺(payment for-here/to-go) (size bread)≻, ≺(size bread) (payment for-here/to-go)≻}	{≺bread size type≻, ≺bread type size≻, ≺size bread type≻, ≺size type bread≻, ≺type size bread≻, ≺type bread size≻}	{≺(bread size type)≻, ≺bread (size type)≻, ≺size (bread type)≻, ≺type (bread size)≻, ≺(bread size) type≻, ≺(bread type) size≻, ≺(size type) bread≻}	{≺(bread size type)≻, ≺(bread size) type≻, ≺type (bread size)≻, ≺(size type) bread≻, ≺bread (size type)≻, ≺(bread type) size≻, ≺size (bread type)≻, ≺bread size type≻, ≺bread type size≻, ≺size bread type≻, ≺size type bread≻, ≺type size bread≻, ≺type bread size≻}
Hasse diagram(s)	 for–here/to–go cheese toasted bread	 room–type rewards# check–out check–out check–in check–in rewards# room–type	 size bread (size bread) sub–dialog 1 payment for–here/to–go (payment for–here/to–go) sub–dialog 2	bread size type	 bread (size type) (size bred type) type (bread size) size (bread type)	bread size type (bread size) (size type) (bread size) (bread size type)

Output	`((C bread (SPE′ cheese toasted) for–here/to–go))`	`((C rewards# check–in check–out room–type)`
		`(C room–type check–in check–out rewards#))`
			`((SPE′ (PE ∗ size bread) (PE ∗ payment for–here/to–go)))`	`((SPE′ bread size type))`	`((I bread size type)`
					`(SPE′ bread (I size type))`
					`(SPE′ size (I bread type))`
					`(SPE′ type (I bread size)))`
						`((PE ∗ bread size type))`

Ratio	(2:1)	(2:2=1:1)	(14:1)	(3! = 6 : 1) $q!$:1	(7:4) $\sum_{p=1}^{q=3} \binom{q}{p}$:1	$\sum_{p=1}^{q=3} p! \times S(q,\ p)$: 1 (13:1)

(b) Continuation through end of space of dialogs which began in table (a) above.

TABLE I: A space of dialogs from fixed (table a, column a) to complete, mixed-initiative dialogs (table b, column k), encompassing a variety of unsolicited reporting, mixed-initiative dialogs, in three representations: enumerated specification (second row), Hasse diagram(s) (third row), and compressed output expression (fourth row).

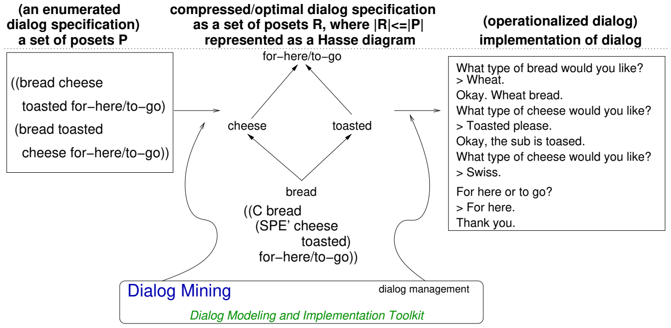

(an enumerated
dialog specification)
a set of posets P
compressed/optimal dialog specification
as a set of posets R, where |R|<=|P|
represented as a Hasse diagram
(operationalized dialog)
implementation of dialog
((bread cheese
toasted for–here/to–go)
(bread toasted
cheese for–here/to–go))
for–here/to–go
cheese
toasted
bread
((C bread
(SPE' cheese
toasted)
for–here/to–go))
What type of bread would you like?
> Wheat.
Okay. Wheat bread.
What type of cheese would you like?
> Toasted please.
Okay, the sub is toased.
What type of cheese would you like?
> Swiss.
For here or to go?
> For here.
Thank you.
Dialog Mining
dialog management
Dialog Modeling and Implementation Toolkit

Fig. 1: Concept of dialog mining (left) contextualized within the broader scope of our approach to automatic dialog system construction. Overview of dialog implementation from initial enumerated specification through dialog mining to an output expression to an interactive dialog system.

and size, and indicate the type of sub desired. Since possible responses to these questions are independent of each other, the dialog designer may wish to permit the customer to communicate the answers in any order or combinations. For instance, some customers may prefer to use a ≺size type bread≻ episode:

SYSTEM: 6 inch or footlong?
USER: Footlong.
SYSTEM: What type of sub is it?
USER: Cold cut combo.
SYSTEM: What type of bread would you like?
USER: Italian herbs and cheese.

Others may prefer a ≺bread size type≻ episode. Still others might prefer to use a ≺(bread size) type≻ episode, where answers to the questions enclosed in parentheses must be communicated in a single utterance (i.e., all at once):

SYSTEM: What type of bread would you like, and is it 6 inch or footlong?
USER: 9-Grain Honey Oat, 6 inch
SYSTEM: What type of sub is it?
USER: Veggie Delight.

To accommodate all dialog completion possibilities we specify this dialog with the Hasse diagram shown in column (k) of Table Ib. The absence of arrows between the bread, size, type, (bread size), (bread type), (size type), and (bread size type) elements indicates that the times at which each of those utterances may be communicated are totally unordered. This specification indicates that answers to the set of questions in the dialog may be communicated in utterances corresponding to all possible set partitions of the set of questions, and using all possible permutations of those partitions. For instance, if the set of questions is {a,b,c}, then the set of all possible partitions is: {{{a},{b},{c}}, {{a,b},{c}}, {{a,c},{b}}, {{a},{b,c}}, {{a,b,c}}}. The set of all permutations of these partitions is: {≺(a b c)≻, ≺(a b) c≻, ≺c (a b)≻, ≺(b c) a≻, ≺a (b c)≻, ≺(a c) b≻, ≺b (a c)≻, ≺a b c≻, ≺a c b≻, ≺b a c≻, ≺b c a≻, ≺c b a≻, ≺c a b≻}. In the Subway example, that set is the set of all six permutations of {bread, size, type} and all permutations of all set partitions of {bread, size, type} or, in other words, all thirteen, possible episodes to complete the dialog shown in shown in column (k) of Table Ib. Notice that a specification of a dialog as a Hasse diagram is a compressed representation capturing its requirements, and the compression is lossless (i.e., the episodes in the enumerated specification may be reconstructed from the diagram).

Granting the user influence over the flow of the dialog increases the number of episodes in its enumerated specification [6]. This Subway-ordering dialog is a *mixed-initiative* dialog [3], [5], [7]. There are multiple degrees of mixed-initiative interaction; the degree considered in this article is called *unsolicited reporting*—an interaction strategy where, in response to a question, at any point in the dialog, the user may provide an unsolicited response to a forthcoming question. When all possible permutations (i.e., orders) of all possible partitions (i.e., combinations) of responses to questions are supported in a dialog, we refer to the dialog as *complete*, mixed-initiative dialog.

B. *A Space of Dialogs*

Table I depicts a space from fixed to complete, mixed-initiative dialogs, encompassing a wide variety of unsolicited reporting, mixed-initiative dialogs. Columns (b)–(j) in Table I contain specifications of dialogs (e.g., for ordering at Subway) that are situated in between the two ends of this space. For instance, consider ordering at Subway where bread and a choice of for-here or to-go must be entered first and last, respectively, but the type of cheese and whether the customer wants the sub toasted or not may be communicated in any order: {≺bread cheese toasted for-here/to-go≻, ≺bread toasted cheese for-here/to-go ≻} (see column (f) of Table Ib). This dialog contains an embedded, mixed-initiative sub-dialog [5]: {≺cheese toasted≻, ≺toasted cheese≻}. Alternatively, consider a dialog for reserving a hotel room where providing a rewards program number and indicating the type of room desired (e.g., king bed) can be communicated either first or last, but the specification of check-in date and check-out date must occur in that order: {≺rewards# check-in check-out room-type≻, ≺room-type check-in check-out rewards#≻}. This dialog contains an embedded, fixed sub-dialog—{≺check-in check-out≻}—and, unlike the prior examples, cannot be captured by a single poset (see column (g) of Table Ib). Lastly, consider the dialog containing two embedded, complete, mixed-initiative sub-dialogs shown in column (h) of Table Ib. Here, the user can pursue the two sub-dialogs in any order, and can specify the atomic elements *within* each sub-dialog in any order (e.g., bread and size in any order, and payment and for-here/to-go in any order), but cannot mix the atomic responses *between* the two sub-dialogs (e.g., the episode ≺size payment bread for-here/to-go≻ is not permitted).

An attractive consequence of specifying dialogs using posets and Hasse diagrams is that the structure of the diagram can be used to design the control structure of the implementation of the dialog. The goal of this paper is to demonstrate how to get from an enumerated specification of a dialog (row two of Table I) to a set of partial orders (row three) and, ultimately, to a compressed specification of the dialog (row four), discussed below (i.e., *dialog mining*; see left side of Fig. 1). In other words, we start with a high-level dialog specification (i.e., a set of episodes) and mine it for a minimum set of posets (i.e., a compressed representation), which entails identifying embedded mixed-initiative sub-dialogs, for purposes of improved dialog implementation.

III. DIALOG MINING ALGORITHM

Extracting a minimal dialog specification from an enumerated specification is a process we refer to as *dialog mining* (see left side of Fig. 1). We generalize this problem to one of finding a minimum set of posets capturing the requirements of a dialog from an enumerated specification of the dialog. Formally, we state the problem as:

INPUT: A set of posets[1] P, all defined over the same set, where the union of the linear extensions from each poset in P is L.

OUTPUT: A minimum set of posets R such that $|R| \leqslant |P|$ and the union of the linear extensions from each poset of R is L.

A *linear extension* of a partial order is a total order that is consistent with the partial order. For instance, $abcd$ is a linear extension of the partial order $\{(a, c), (b, d)\}$. Intuitively, the goal of the mining algorithm is to identify the solicitations in the dialog specification whose responses are unordered (i.e., can be communicated at any time) or, in other words, all the parts of the dialog involving mixed initiative. Specifically, what we are attempting to mine is the patterns shown in the first row of Table II, which each consist of a set of episodes with particular properties. For instance, if we find the set of episodes in the cell at the sixth column, first row of Table II, we compress that set of episodes by representing them with the expression (`PE` $*$ `a b c`), shown in the second row of Table II. While beyond the scope of this paper, which focuses on the mining aspect, the first five patterns shown in the top row of Table II correspond to five concepts from programming languages [8]. The main idea is that if we want to support a subset of episodes in the dialog implementation that matches one of these patterns, we can advantageously view the support for those episodes through the lens of the language-based concept to which we associate it. These concepts from programming languages (and combinations of them) help specify dialogs between the fixed and complete, mixed-initiative ends of the space of dialogs depicted in Table I and, thus, help bring structure to this space [9]. For purposes of this paper, it is sufficient to understand that each of these patterns represents a particular type of compression of the input.

We designed a recursive, heuristic-based algorithm to address this problem. Table III gives thirteen tests cases that we use to demonstrate the heuristic nature of our algorithm. We use *S-list* notation [10] to describe the input and output of our algorithm, because our implementation of the algorithm is in a LISP-like language. However, the lists are only used to represent sets (i.e., they never contain duplicate elements). The input to the algorithm is an enumerated specification we refer to as m-episodes. They are given as a list of LISP lists [11], where each inner list corresponds to an m-episode, where m is the number of atomic elements in the episode (e.g., ≺a b c≻), not necessarily the length of the episode (e.g., the episode ≺(a b c)≻ is of length one, but has three atomic elements). Similarly, the output is a set of posets represented as a list of LISP lists, where each inner list represents a poset. Each inner list in the output list begins with one of the five mnemonics that are used to compress the set of input episodes (i.e., `C`, `I`, `PFAN`$*$, `SPE`$'$, `PE`$*$; see second row of Table II).

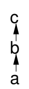
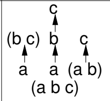
Input:	≺a b c≻	≺(a b c)≻	≺(a b c)≻, ≺a (b c)≻, ≺(a b) c≻, ≺a b c≻	≺a b c≻, ≺a c b≻, ≺b a c≻, ≺b c a≻, ≺c b a≻, ≺c a b≻	≺(a b c)≻, ≺(a b) c≻, ≺c (a b)≻, ≺(b c) a≻, ≺a (b c)≻, ≺(a c) b≻, ≺b (a c)≻, ≺a b c≻, ≺a c b≻, ≺b a c≻, ≺b c a≻, ≺c b a≻, ≺c a b≻	≺a b c≻, ≺a c b≻
Output:	`(C a b c)`	`(I a b c)`	`(PFAN` $*$ `a b c)`	`(SPE`$'$ `a b c)`	`(PE` $*$ `a b c)`	`(C a (SPE`$'$ `b c))`
Hasse Diagram(s):		(a b c)		a b c	a (b c) c (a b) (a c) (a b c) b	
Concept:	**C**urrying	**I**nterpretation	**P**artial **F**unction **A**pplication	**S**ingle argument **P**artial **E**valuation	**P**artial **E**valuation	N/A
Semantics:	a sequences of (3) utterances	multiple (3) responses per utterance	all combinations of `a b c` in that order	all (6) permutations of `a b c`	all (13) permutations and combinations of `a b c`	sub-dialog
Compression Ratio:	1:1	1:1	2^{q-1}:1 ($2^{3-1=2} = 4$:1)	$q!$:1 ($3! = 6 : 1$)	$\sum_{p=1}^{q=3} p! \times S(q,\ p) : 1$ (13:1)	N/A

TABLE II: Patterns for which we mine.

It is helpful to think of the algorithm as processing a set of m-episodes in parallel from left to right as presented in Table III. If all possible permutations (i.e., orders) of all possible partitions (i.e., combinations) of atomic elements are given in the m-episodes (i.e., a *complete*, mixed-initiative dialog), we compress them into a poset set of one poset element (e.g., case 1 in Table III). If only all possible permutations of the atomic elements are given, we also compress them into a set of one poset element (e.g., case 2 in Table III). If neither of these cases is satisfied, the algorithm checks if the first element of each input m-episode is equal to each other. If so (e.g., case 3 in Table III), the algorithm partially constructs the output set of posets, removes the first element of each input m-episode, and recursively applies itself to the remaining input (m-1)-episodes, while maintaining the partially-constructed set of posets.

For purposes of overall algorithm explanation, it is non-essential to distinguish between all possible permutations (i.e., orders) of all possible partitions (i.e., combinations) of atomic elements and just all possible permutations of the atomic elements. However, since the exposition of the latter is simpler, we use the pattern of all possible permutations of the atomic elements in the remainder of our illustrative examples.

If the first element of each input m-episode is not equal to each other, the algorithm computes p, which is the inverse factorial of s, the number of episodes input. For example, in case 4 in Table III, because s = 2 and m = 3, p is computed to be 2 because p!=2!=s=2. After computing p, the algorithm checks for all possible p permutations of the atomic elements at the beginning of each input m-episode. If found, (e.g., case 4 in Table III), the algorithm partially constructs the output set of posets and removes all of the p-permutations from the beginning of the input m-episodes. The algorithm is then applied recursively to the remaining (m-p)-episodes, again, maintaining the partially-constructed output set of posets. The recursion in this algorithm terminates when the final element in each input m-episode is processed.

---

[1] While each episode in the input set is a totally ordered set, we refer to the input as a set of posets since any totally ordered set is also a partially ordered set.

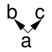
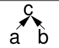
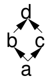
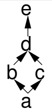
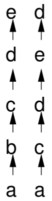
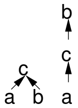
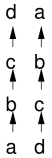
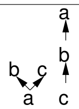
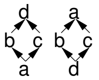
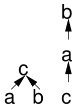
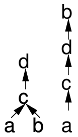
#	**Input** (m-episodes)	**Output**	**Hasse Diagram(s)**	**Compression Ratio**
1	((│ a b │) (│ b a │) (│ (a b) │))	`(((PE* a b)))`	(a b) a b	(3:1)
2	((│ a b │) (│ b a │))	`(((SPE' a b)))`	a b	(2:1)
3	((a │ b c │) (a │ c b │))	`((C a` `(SPE' b c)))`		(2:1)
4	((│ a b │ c) (│ b a │ c))	`((C (SPE' a b) c))`		(2:1)
5	((a │ b c │ d) (a │ c b │ d))	`((C a` `(SPE' b c) d))`		(1:1)
6	((a │ b c │ d e │) (a │ c b │ d e │))	`((C a` `(SPE' b c) d e))`		(2:1)
7	((a │ b c │ d e │) (a │ c b │ e d │))	`((C a b c d e)` `(C a c b e d))`		(2:2=1:1)
8	((│ a b │ c) (│ b a │ c) (a c b)) ((a │ b c │) (a │ c b │) (b a c))	`((C (SPE' a b) c)` `(C a c b))`		(3:2)
9	((a │ b c │ d) (d │ c b │ a))	`((C a b c d)` `(C d c b a))`		(2:2=1:1)
10	((a │ b c │) (a │ c b │) (c b a))	`((C a (SPE' b c))` `(C c b a))`		(3:2)
11	((a │ b c │ d) (d │ c b │ a) (d │ b c │ a) (a │ c b │ d))	`((C a (SPE' b c) d)` `(C d (SPE' c b) a))`		(4:2=2:1)
12	((│ a b │ c) (│ b a │ c) (c a b))	`((C (SPE' a b) c)` `(C c a b))`		(3:2)
13	((a b c d) (a c d b) (b a c d))	`((C (SPE' a b) c d)` `(C a c d b))`		(3:2)

TABLE III: Test cases used in our explanation of our dialog mining algorithm, and especially its heuristic nature. The vertical rules in the cells of the second column serve to make the instance of the pattern for which we mine salient. The horizontal rules in the cells of the second column illustrate how the algorithm groups episodes in the input using the heuristic.

Cases 1–6 in Table III yield compression in the output set of posets (see column labeled 'Compression Ratio'). In cases 7 and 9, after running the algorithm, each m-episode in the input corresponds to a poset (or a totally order set in this case) yielding no compression. Note that the cell at the input column of case 8 contains two sets of the same m-episodes, just listed in different orders. Our algorithm computes the same set of posets for each demonstrating that our algorithm is not sensitive to the presentation order of each m-episode in the input list.

#expr	**Total**	**w/ Com.**	**%tage**	**w/o Com.**	**%tage**
1	46	33	72 %	13	28 %
2	435	372	86 %	63	14 %
3	1,705	1,558	91 %	147	9 %
4	2,977	2,800	94 %	177	6 %
5	2,291	2,184	95 %	107	5 %
6	688	662	96 %	26	4 %
7	49	49	100 %	0	0 %
8	0	N/A	N/A	N/A	N/A
9	0	N/A	N/A	N/A	N/A
10	0	N/A	N/A	N/A	N/A
11	0	N/A	N/A	N/A	N/A
12	0	N/A	N/A	N/A	N/A
13	0	N/A	N/A	N/A	N/A
Totals:	8,191	7,658	**93 %**	**533**	7 %

TABLE IV: Frequencies of dialogs that can be represented with 1–13 expressions, with and without compression.

Legend: **#expr** = 'number of expressions'; **w/ Com.** = 'with compression'; **w/o Com.** = 'without compression'; **%tage** = 'percentage.'

If all p-permutations are not found at the beginning of the m-episodes, the algorithm groups the episodes by first element (e.g., cases 9–13 in Table III) and recursively processes each group independently. The primary goal/theme of dialog mining is to identify as many embedded complete, mixed-initiative sub-dialogs in an enumerated specification since they can be advantageously handled in the implementation by the program transformation partial evaluation [12]. This algorithm achieves the process that is referred to as *layering* [13]. We implemented this dialog mining algorithm in Racket, a dialect of the LISP programming language. The miner runs in *DrRacket*[2] (version 6.1.1) with the language set to 'Determine language from source.'

IV. EVALUATION

Our algorithm is sound in that it always finds a set of posets whose linear extensions constitute the input set (i.e., it never returns a wrong answer), but not complete. If it cannot mine a minimum set of posets, it returns the input set of linear extensions as the output, which always describes the linear extensions of the input. However, it is incomplete in that it does not always mine the minimum number of posets (i.e., minimum specification).

Another way to evaluate our mining algorithm is to measure the fraction of input episodes it compresses across a wide range of dialogs. To do this we must first characterize the input space of dialogs. Intuitively, since the enumerated specification of a **c**omplete, **m**ixed-**i**nitiative dialog, denoted as D_{cmi_q}, contains the maximum number of episodes possible, given q,

---

[2] http://racket-lang.org/

Com. Ratio	**% Com.**	**Ind. Freq.**	**Cum. Freq.**	**Cum. %age**
13 : 1	92 %	1	1	0 %
6 : 1	83 %	4	4	0 %
8 : 2 4 : 1	75 %	8 7	15	0 %
7 : 2	71 %	22	22	0 %
10 : 3	70 %	8	8	0 %
12 : 4 9 : 3 6 : 2 3 : 1	66 %	2 29 27 6	64	1 %
11 : 4	**63 %**	**17**	**17**	**2 %**
8 : 3	62 %	83	83	3 %
10 : 4 5 : 2	60 %	49 78	127	4 %
12 : 5	58 %	8	8	4 %
7 : 3	57 %	179	179	6 %
9 : 4	55 %	160	160	8 %
11 : 5	54 %	29	29	9 %
12 : 6 10 : 5 8 : 4 6 : 3 4 : 2 2 : 1	**50 %**	3 118 378 357 104 15	**975**	**21 %**
11 : 6	45 %	27	27	21 %
9 : 5	44 %	313	313	25 %
7 : 4	42 %	696	696	33 %
10 : 6 5 : 3	40 %	94 475	569	40 %
8 : 5	37 %	586	586	47 %
11 : 7	36 %	5	5	47 %
9 : 6 6 : 4 3 : 2	33 %	193 871 133	1,197	62 %
10 : 7	30 %	17	17	62 %
7 : 5	28 %	699	699	71 %
8 : 6 4 : 3	**25 %**	225 427	652	**79 %**
9 : 7	22 %	20	20	79 %
5 : 4	20 %	627	627	87 %
6 : 5	16 %	431	431	92 %
7 : 6	14 %	120	120	93 %
8 : 7	12 %	7	7	93 %
6 : 6 5 : 5 4 : 4 3 : 3 2 : 2 1 : 1	0 %	26 107 177 147 63 13	**533**	**93 %**
Total (= $\lvert\mathcal{U}_3\rvert$):		8,191	8,191	

TABLE V: Frequency of dialogs in $\mathcal{U}_3$ compressed to discrete compression percentages observed.

Legend: **Com. Ratio** = 'compression ratio'; **Ind. Freq.** = 'individual frequencies'; **Cum. Freq.** = 'cumulative frequency'; **Cum. %tage** = 'cumulative percentage'.

the fixed number of questions posed in the dialog, the set of all possible subsets, save for the empty set, of the set of all episodes in $D_{cmi\,q}$ represents a space of all dialogs possible given q in our model. There is a combinatorial explosion in the number of dialogs possible between the fixed and complete, mixed-initiative ends of the space of dialogs discussed here. Specifically, the number of dialogs possible in this space is $\mathcal{U}_q = 2^{|D_{cmi\,q}|} - 1 = \sum_{r=1}^{|D_{cmi\,q}|} \binom{|D_{cmi\,q}|}{r}$ (i.e., $|D_{cmi\,q}|$ choose r, or all possible subsets, except for the empty set, of all episodes in a dialog $D_{cmi\,q}$). To enumerate the dialogs in this space, for subsequently mining them, we must first capture the size of the set $D_{cmi\,q}$.

Let $s(m)$ be the set of all partitions of a set of size m into non-empty subsets, where m is a positive integer (e.g., $s(3) = \{\{\{a\},\{b\},\{c\}\}, \{\{a,b\},\{c\}\}, \{\{a,c\},\{b\}\}, \{\{a\},\{b,c\}\}, \{\{a,b,c\}\}\}$), and $s(m, n)$ be the set of all partitions of a set of size m into exactly n non-empty subsets, where n is a positive integer and $n \leqslant m$. The *Bell number* of a set of size m is $B(m) = |s(m)|$ (e.g., $B(3) = 5$). The *Stirling number* of a set of size m is $S(m, n) = |s(m, n)|$. It follows that $B(m) = \sum_{n=1}^{m} S(m, n)$. Since an enumerated specification of a complete, mixed-initiative dialog contains episodes corresponding to all possible permutations of all possible partitions of the set of questions in the dialog, we define its size, $|D_{cmi\,q}|$, as the total function, $\mathbb{N} \rightarrow \mathbb{N}$, equal to $\sum_{p=1}^{q} p! \times S(q,\ p)$, which given q, the number of questions posed in a dialog, computes the total number of episodes therein. Using this closed formula for $|D_{cmi\,q}|$, we observe that $|D_{cmi\,1}| = 1$, $|D_{cmi\,2}| = 3$, $|D_{cmi\,3}| = 13$, $|D_{cmi\,4}| = 75$, and $|D_{cmi\,5}| = 541$. Computing the number of all subsets (minus the empty set) of these, we see that $\mathcal{U}_1 = 1$, $\mathcal{U}_2 = 7$, $\mathcal{U}_3 = 8{,}191$, $\mathcal{U}_4 = 3.7 \times 10^{21}$. Clearly, we cannot enumerate all of the dialogs in $\mathcal{U}_4$. Therefore, we generated (i.e., enumerated) all of the dialogs in $\mathcal{U}_3$ as a synthetic data set through which to run and evaluate our algorithm as a dialog simulation. Thus, we compute and analyze compression ratios of the dialogs in $\mathcal{U}_3$. The use of simulation for evaluation of dialog systems is common [2], [14], [15], [16].

The fifth and final row of Table I gives the compression ratio for the cases shown there. The ratio given in parentheses is the compression ratio for the particular (test) case shown while the unparenthesized ratio gives the more general form, if applicable. For instance, in column (k) thirteen episodes are compressed into one output expression (in this case, poset) and, thus, we see the compression ratio as (13:1). Given q, the number of questions per episode in a dialog specification, the general compression ratio is $\sum_{p=1}^{q=3} p! \times S(q,\ p) : 1$. In column (a) there is no compression and, thus, there is a one-to-one correspondence between the number of episodes in the enumerated specification and the number of compression expressions (in this case, posets).

The dialogs in any set $\mathcal{U}_q$ whose episodes can be compressed to one output expression include those that can be represented only by the `(PFAN * a b c)`, `(SPE′ a b c)`, `(PE * a b c)` patterns in Table II; see the last row for those compression ratios which all end in :1. Patterns C and I do not qualify because they exhibit no compression (i.e., 1:1 compression ratio; see last row of Table II).

Of the total dialogs in $\mathcal{U}_q$, $2^{|D_{cmi\,q}|} - 2q! - 3$ dialogs cannot be compressed to only a single output expression containing only one concept mnemonic (e.g., dialogs c, d, f, h, and j in Table I); we refer to the sub-space of $\mathcal{U}_q$ containing these dialogs as Δ_q. Each dialog in Δ_q is represented in the output of our algorithm with more than one poset (e.g., dialog g in Table Ib) or with sub-dialogs through nesting [17] (e.g., dialogs f and h in Table Ib), or both (e.g., dialogs c, d, and j in Table I). The sub-space Δ_3 of $\mathcal{U}_3$ contains 8,176 dialogs.

We ran all of the dialogs in $\mathcal{U}_3$ through our dialog mining system. Our results are given in Tables IV and V. Table IV shows the distribution of the 8,191 dialogs in $\mathcal{U}_3$ mined across the number of expressions in the output. For instance, a dialog in $\mathcal{U}_3$ that contains exactly five episodes that our miner compressed to three expressions is counted in the total 1,705 dialogs and among the 1,558 dialogs with compression in row 3 of Table IV. However, a dialog in $\mathcal{U}_3$ that contains exactly three episodes and could not be compressed by our miner is counted in the total 1,705 dialogs, but counted among the 147 dialogs without compression in row 3. Table IV ranges from 1 to 13 expressions because given a dialog with three questions, there are thirteen possible episodes. Even though there are dialogs in $\mathcal{U}_3$ containing between eight and thirteen episodes, our algorithm is able to compress all of them to a number of episodes between one and seven. This, however, does not mean that our algorithm is able to compress 100 % of the dialogs in $\mathcal{U}_3$. The last row of Table IV indicates that the algorithm is unable to compress 7 % of the dialogs in $\mathcal{U}_3$. Those that it was unable to compress started with a number of episodes between one and six, and the miner output the same number of episodes input. However, our algorithm did compress 93 % of the dialogs in $\mathcal{U}_3$. Table V drills down into the dialogs that the algorithm did compress to examine the magnitude of the compression in each dialog compressed.

Table V enumerates each discrete percent compression we observed across all dialogs in $\mathcal{U}_3$. For instance, a 63 % compression percentage is indicated by a 11:4 compression ratio (i.e., 7 of the original 11 episodes were compressed; 7/11 = 63 %). Table V also provides the frequency of dialogs at each compression ratio observed as well as the cumulative frequency of dialogs revealed at each percent compression. For instance, there were 975 dialogs compressed to 50 % of the number of episodes input and those 975 dialogs were distributed among six different compression ratios. Table V reveals that a majority of the 533 uncompressed dialogs noted in Table IV started (and ended) with between three, four, or five episodes. The column labeled 'Cum. %age' (i.e., cumulative percentage) of Table V shows that 21 % of dialogs in $\mathcal{U}_3$ are each compressed to a level of 50 % or more, and 79 % of them are each compressed to a level of 25 % or more. We expect an analysis of $\mathcal{U}_4$ to reveal even better results because as the number of questions posed in a dialog increases (which also yields a much larger number of episodes possible in the space), the opportunities for compression increase. Due to the combinatorial explosion in the number of dialogs in $\mathcal{U}_4$ we are unable to enumerate them in preparation for mining them with our algorithm.

V. RELATED RESEARCH

Data mining algorithms have been used for a variety of purposes by the mixed-initiative dialog research community. One approach entails mining data from the web, including web search query logs [18], web content [15], [19], and question-and-answer fora [20], to automatically acquire domain knowledge for use in dialog specification task structures used in a knowledge-based dialog management component of a (automatically created) dialog system [2]. An alternate approach involves mining dialogs from observed conversions, including doctor-patient dialogs [21], or spoken dialog corpora [22], often for dialog modeling or analysis rather than system development. Other approaches include using machine learning (e.g., reinforcement learning) to mine dialog strategies from observed human-computer dialogs [23]. We view our work as complementary to these research efforts. For instance, any of the data that Feng, Hakkani-Tür, Di Fabbrizio, Gilbert, and Beutnagel [19] mine from the web that can be interpreted as sequences of dialog steps can be run through our algorithm to identifying opportunities to mix initiative in preparation for (automatic) dialog system creation. More generally, Glass and Seneff state that they "see several different ways in which such flexible reconfiguration will become feasible in the near future. Perhaps the most critical is the initial preparation of a new domain, where available on-line databases will be the catalyst for defining the vocabulary and language models of the domain, *as well as the nature of the dialogue interaction needed to guide the user through the information space*"[3] [1]. Our miner identifies the *nature of the dialogue interaction needed to guide the user through the information space*, in this case partially ordered sets corresponding to mixing initiative, and, thus, dovetails with these efforts, especially with respect to initially and dynamically (re-)configuring dialog systems based on the availability of (new) domain information [1], [15], [19], [24].

VI. DISCUSSION

Dialog mining is one aspect of our approach to automate the implementation of a human-computer dialog from a high-level specification of it, in a similar vein as [15], [19], [25]. While the details of dialog system construction are beyond the scope of this paper, we make some remarks to convey, in a larger context, the motivation for and purpose of mining the parts of dialog specification where initiative is mixed, as well as what we do with the resulting mined expression(s), which represent(s) a compressed specification of a dialog.

The overall idea is that a dialog implementation resulting from an optimal output expression specifying the dialog is more efficient than one generated from a sub-optimal expression. An optimal expression is one which captures the maximum number of the patterns described above from the enumerated specification or, in other words, one which exhibits the highest level of compression.

A primary advantage to our approach to dialog implementation is that to support all of the episodes in the enumerated specification of a dialog, we need not explicitly model all of them in the control flow of the implementation; we only need to model each output expression, which, as demonstrated above, is less than the number of episodes in the enumerated specification 93 % of the time (for q=3). Thus, in addition to its use in the evaluation of our mining algorithm, the number of independent output expressions is also an implementation-neutral method of quantifying control complexity of a dialog implementation in this model. Intuitively, the column labeled '#expr' in Table IV captures the number of independent control flows that must be explicitly enumerated in the implementation of each dialog counted in the column labeled 'w/ Com.'

---

[3] Our emphasis.

Composing the mining (Fig. 1, left) and management (right) components results in a dialog modeling and implementation toolkit. We start with an enumerated dialog specification (i.e., a set of episodes) and mine it for a compressed representation of the dialog as a set of partially ordered sets that captures the requirements of the dialog (see transition from the left to the center of Fig. 1). From that intermediate representation we automatically generate a dialog system capable of managing (the progressive interaction of) the dialog (transition from the center to the right side of Fig. 1). Fig. 1 situates dialog mining within the scope of our broader research on the specification and implementation of mixed-initiative dialogs.

In conclusion, there are three inter-related aspects of our work relevant to this article: i) specifying dialogs by the completion paths through the dialog, ii) identifying the mixing of initiative between these paths, and iii) supporting that mixing of initiative between those paths in the implementation using partial evaluation.This article focuses on aspect (ii).

REFERENCES

[1] J. Glass and S. Seneff, "Flexible and personalizable mixed-initiative dialogue systems," in *Proceedings of the North American Chapter of the Association for Computational Linguistics (ACL): Human Language Technologies (NAACL-HLT) Workshop on Research Directions in Dialogue Processing*. Stroudsburg, PA: Association for Computational Linguistics, 2003, pp. 19–21.

[2] C. Lee, S. Jung, K. Kim, D. Lee, and G. Lee, "Recent approaches to dialog management for spoken dialog systems," *Journal of Computing Science and Engineering*, vol. 4, no. 1, pp. 1–22, 2010.

[3] M. Walker and S. Whittaker, "Mixed-initiative in dialogue: An investigation into discourse segmentation," in *Proceedings of the Twenty-eighth Annual Meeting on Association for Computational Linguistics (ACL)*. Stroudsburg, PA: Association for Computational Linguistics, 1990, pp. 70–78.

[4] S. Hamidi, P. Andritsos, and S. Liaskos, "Constructing adaptive configuration dialogs using crowd data," in *Proceedings of the Twenty-ninth ACM/IEEE International Conference on Automated Software Engineering (ASE)*. New York, NY: ACM Press, 2014, pp. 485–490.

[5] J. Allen, "Mixed-initiative interaction," *IEEE Intelligent Systems*, vol. 14, no. 5, pp. 14–16, 1999.

[6] J. Chu-Carroll, "MIMIC: An adaptive mixed initiative spoken dialogue system for information queries." in *Proceedings of the Sixth Conference on Applied Natural Language Processing (ANLC)*. Stroudsburg, PA: Association for Computational Linguistics, 2000, pp. 97–104.

[7] V. Zue and J. Glass, "Conversational interfaces: advances and challenges," *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1166–1180, 2000.

[8] M. Scott, *Programming Language Pragmatics*, 3rd ed. Amsterdam: Morgan Kaufmann, 2009.

[9] S. Perugini and J. Buck, "A language-based model for specifying and staging mixed-initiative dialogs," in *Proceedings of the Eighth International ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*, J. C. Campos and A. Schmidt, Eds. New York, NY: ACM Press, 2016, pp. 204–216.

[10] D. Friedman and M. Wand, *Essentials of Programming Languages*, 3rd ed. Cambridge, MA: MIT Press, 2008.

[11] C. Queinnec, *LISP in Small Pieces*. Cambridge, UK: Cambridge University Press, 1996.

[12] N. Jones, "An introduction to partial evaluation," *ACM Computing Surveys*, vol. 28, no. 3, pp. 480–503, 1996.

[13] N. Ramakrishnan, R. Capra, and M. Pérez-Quiñones, "Mixed-Initiative Interaction = Mixed Computation," in *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, P. Thiemann, Ed. New York, NY: ACM Press, 2002, pp. 119–130, also appears in *ACM SIGPLAN Notices*, 37(3), 2002.

[14] T. Misu, K. Georgila, A. Leuski, and D. Traum, "Reinforcement learning of question-answering dialogue policies for virtual museum guides," in *Proceedings of the Thirteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Stroudsburg, PA: Association for Computational Linguistics, 2012, pp. 84–93.

[15] J. Polifroni, G. Chung, and S. Seneff, "Towards the automatic generation of mixed-initiative dialogue systems from web content," in *Proceedings of the Eighth European Conference on Speech Communication and Technology (EUROSPEECH)*. International Speech Communication Association, 2003, pp. 193–196.

[16] R. Inouye, "Minimizing the length of non-mixed initiative dialogs," in *Proceedings of the Association for Computational Linguistics (ACL) Workshop on Student Research*. Stroudsburg, PA: Association for Computational Linguistics, 2004.

[17] R. Capra, M. Narayan, S. Perugini, N. Ramakrishnan, and M. Pérez-Quiñones, "The staging transformation approach to mixing initiative," in *Working Notes of the IJCAI 2003 Workshop on Mixed-Initiative Intelligent Systems*, G. Tecuci, Ed. Menlo Park, CA: AAAI/MIT Press, 2003, pp. 23–29.

[18] D. Hakkani-Tür, G. Tür, and A. Celikyilmaz, "Mining search query logs for spoken language understanding," in *Proceedings of the North American Chapter of the Association for Computational Linguistics (ACL): Human Language Technologies (NAACL-HLT) Workshop on Future Directions and Needs in the Spoken Dialog Community: Tools and Data*. Stroudsburg, PA: Association for Computational Linguistics, 2009, pp. 37–40.

[19] J. Feng, D. Hakkani-Tür, G. D. Fabbrizio, M. Gilbert, and M. Beutnagel, "Webtalk: Towards automatically building spoken dialog systems through mining websites," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Los Alamitos, CA: IEEE Computer Society Press, 2006, pp. 573–576.

[20] W. Wong, L. Cavedon, J. Thangarajah, and L. Padgham, "Mixed-initiative conversational system using question-answer pairs mined from the web," in *Proceedings of the Twenty-first ACM International Conference on Information and Knowledge Management (CIKM)*. New York, NY: ACM Press, 2012, pp. 2707–2709.

[21] K. Mase, Y. Sawamoto, Y. Koyama, T. Suzuki, and K. Katsuyama, "Interaction pattern and motif mining method for doctor-patient multimodal dialog analysis," in *Proceedings of the International Conference on Multimodal Interfaces and Workshop on Machine Learning for Multimodal Interaction (ICMI-MLMI) Workshop on Multimodal Sensor-Based Systems and Mobile Phones for Social Computing*. New York, NY: ACM Press, 2009, pp. 6:1–6:4.

[22] F. Bechet, G. Riccardi, and D. Hakkani-Tür, "Mining spoken dialogue corpora for system evaluation and modeling," in *Proceedings of the Association for Computational Linguistics (ACL) Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Stroudsburg, PA: Association for Computational Linguistics, 2004, pp. 134–141.

[23] M. English and P. Heeman, "Learning mixed-initiative dialog strategies by using reinforcement learning on both conversants," in *Proceedings of the North American Chapter of the Association for Computational Linguistics (ACL): Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (NAACL-HLT/EMNLP)*. Stroudsburg, PA: Association for Computational Linguistics, 2005, pp. 1011–1018.

[24] X. Yao, E. Tosch, G. Chen, E. Nouri, R. Artstein, A. Leuski, K. Sagae, and D. Traum, "Creating conversational characters using question generation tools," *Dialogue and Discourse*, vol. 3, no. 2, pp. 125–146, 2012.

[25] S. Kronenberg and P. Regel-Brietzman, "Bridging the gap between mixed-initiative dialogs and reusable sub-dialogs," in *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 276–279.