Constructing Adaptive Configuration Dialogs using Crowd Data

Saeideh Hamidi
School of Information Technology
York University
Toronto, Canada
hamidi@yorku.ca

Periklis Andritsos
Faculty of Business and Economics
University of Lausanne
Lausanne, Switzerland
periklis.andritsos@unil.ch

Sotirios Liaskos
School of Information Technology
York University
Toronto, Canada
liaskos@yorku.ca

ABSTRACT

As modern software systems grow in size and complexity so do their configuration possibilities. Users are easy to be confused and overwhelmed by the amount of choices they need to make in order to fit their systems to their exact needs. We propose a method to construct adaptive configuration elicitation dialogs through utilizing crowd wisdom. A set of configuration preferences in the form of association rules is first mined from a crowd configuration data set. Possible configuration elicitation dialogs are then modeled through a Markov Decision Process (MDP). Association rules are used to inform the model about configuration decisions that can be automatically inferred from knowledge already elicited earlier in the dialog. This way, an MDP solver can search for elicitation strategies which maximize the expected amount of automated decisions, reducing thereby elicitation effort and increasing user confidence of the result. The method is applied to the privacy configuration of Facebook.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement—*customization and configuration*; H.4.2 [**Information Systems Applications**]: Types of Systems—*decision support*

General Terms

Management

Keywords

Software Customization, Crowdsourcing, Markov Decision Processes, Association Rules Mining, Facebook

1. INTRODUCTION

Modern software systems exhibit functionalities in amounts and complexities that have never been seen before. As systems mature and new challenges and opportunities emerge, more features are developed and added to an already large and complex set of existing ones. Users interested to fully exploit the power of their software are invited to choose the functions they need and *configure* them in a way that perfectly aligns with their goals, preferences and capabilities [8]. But as the amount and complexity of functions grows, so does the space of configuration possibilities. Thus, when attempting to fit the functionality of their systems to their unique needs, users, especially novice ones, have to deal with an increasingly mystifying and overwhelming task.

In this paper, we propose a way to reduce configuration effort for users by building adaptive configuration dialogs via utilizing configuration preferences of a crowd. The method is based on combining *association rule mining* with *Markov Decision Processes (MDP)*. Rules that describe combinations of configurations that frequently coincide are first mined from the crowd configuration data. Then, an MDP model is constructed in which states correspond to the state of our knowledge of the preferred configuration, and actions are configuration questions posed to the users in order to learn their preferred option for a given variable. User response to the action naturally triggers transition from a less informed to a more informed state; but the association rules help us multiply the amount of knowledge we gain with each answer, effectively skipping unnecessary questions. By solving the MDP through rewarding shorter paths towards more configuration knowledge, we are able to minimize the length of the sequence of questions that are required for a user to have a complete configuration. We apply our technique on data collected from the popular social networking system Facebook where we show that configuration effort is substantially reduced on average. Moreover, in performance experimentation we show that the solver can handle a useful size of configuration spaces within practical computation time.

Our contributions include: (a) a systematic way of developing adaptive configuration elicitation dialogs, (b) minimal, on average, length of such dialogs, and (c) a model based on evidence from existing configurations, potentially from expert users.

The paper is organized as follows. In Section 2 we present an overview of the problem. In Section 3 we discuss the mining of crowd configuration preferences and in Section 4 we offer the details of our solution. We describe our evaluation in Section 5, we survey related work in Section 6 and we conclude in Section 7.

ASE'14, September 15-19, 2014, Vasteras, Sweden.

http://dx.doi.org/10.1145/2642937.2642960.

2. MOTIVATION AND BASIC ASSUMPTIONS

Configuring modern software systems is a process of assigning values to a number of parameters that the designers of the software system avail to users within various configuration facilities, such as "Options", and "Preference" screens or configuration files. In more abstract terms, a configuration problem consists of a set $\mathcal{V}$ of configuration *variables* V_i each drawing values o_j^i (*options*) from a set $\mathcal{O}_{V_i}$.

Consider a very simple configuration problem from a hypothetical email client program. Among its many configuration variables (e.g. whether to use HTML, what connection security to apply, whether to include a signature in each email, etc.), assume that our email program allows the user to also adjust *font size*, *icon size* (for buttons such as "New E-Mail" and "Reply") and whether to also *display text* under the icons. We denote these variables as V_{fnt}, V_{ico} and V_{disp}, respectively. The user can adjust the variables to take values from the sets: {*large*, *medium*, *small*}, {*large icon*, *small icon*} and {*yes*, *no*}, respectively. Thus, using our above formalization the domain of each variable is as follows:

$\mathcal{O}_{V_{fnt}} = \{o_{lg}^{fnt}, o_{med}^{fnt}, o_{sm}^{fnt}\}$
$\mathcal{O}_{V_{ico}} = \{o_{lgi}^{ico}, o_{smi}^{ico}\}$
$\mathcal{O}_{V_{disp}} = \{o_{yes}^{disp}, o_{no}^{disp}\}$

Note that although we assume discrete and finite domains for the variables, continuous variables that often occur in configuration problems can be appropriately discretized to fit our framework, by defining intervals and representative values therein.

A *configuration decision* is the (human/non-automated) act of thinking and deciding what option is more suitable or preferred for a particular variable. In our case, in order to completely solve the configuration problem, our email client user would need to make three configuration decisions, thinking and deciding about each variable separately.

Nevertheless, we may also have some *crowd configuration preferences* in form or rules available to us. Such rules would tell us that if the user decides to adjust some of her configuration variables in a specific way, then she also wants to adjust some other configuration variables in another way. The general form of an association rule is:

$(V_{l_1} = o_{\cdot}^{l_1}) \wedge (V_{l_2} = o_{\cdot}^{l_2}) \wedge \ldots \longrightarrow (V_{r_1} = o_{\cdot}^{r_1}) \wedge (V_{r_2} = o_{\cdot}^{r_2}) \wedge \ldots$

where V_{l_i} and V_{r_i} are respectively variables on the left-hand and right-hand side of the rule, and $o_{\cdot}^{l_i}$ and $o_{\cdot}^{r_i}$ are arbitrary values drawn from the corresponding domains.

Back to our email client example, we may be aware that if a user prefers to have large font size, she also likes to have large icon size. Furthermore, we know that if she likes large icon size then she also prefers text to not be displayed under the icon, apparently to save space. Here is how we can write those rules more formally:

$(V_{fnt} = o_{lg}^{fnt}) \longrightarrow (V_{ico} = o_{lgi}^{ico})$
$(V_{ico} = o_{lgi}^{ico}) \longrightarrow (V_{disp} = o_{no}^{disp})$

Notice now how this knowledge, whenever available, can help us reduce configuration effort. Recall that in order to have the complete set configured, the user would normally need to make three separate decisions. However, if we assume presence of the association rules we know that some choices in some variables help us infer choices in other variables, saving us from making the corresponding configuration decisions. Given this, it is easy to see that there are more efficient and less efficient *orderings* by which the configuration decisions can be made. Given the above two configuration rules, if the user decides about text display first, then icon size, and finally font size, she will always have to make three decisions. If, on the contrary, she starts by deciding what font size she wants, then, depending on her decision on that, there is a chance that the other two decisions need not be made but are taken care of by the rules.

More generally, given a set of configuration variables to be decided upon, and a set of crowd preference rules among them, what is the optimal sequence in which potential decisions can be ordered, so that the smallest number of configuration decisions is eventually made on average? Knowing how to compute such a sequence of decisions could be useful for constructing adaptive configuration dialogs in which users are directed towards making more influential decisions first, allowing thereby a larger subset of variables to be configured automatically without the need for a human decision. Such adaptive configuration dialogs can be used when the entire configuration of some aspect of the system needs to be addressed, e.g. during installation time. In the rest of the paper we describe a system that makes this possible.

3. MINING CROWD CONFIGURATION PREFERENCES

Crowd configuration preferences in our technique come in the form of association rules [1], which are mined from a data set with configurations of other users. Association rules are expressed in the form $(X \longrightarrow Y)(c, s)$, where X and Y are sets of items that often appear together. The rule states when X occur, Y occur with certain probability c, which is referred to as *confidence* and is used as a measure of significance of the rule. The level of *support* s is another significance measure and denotes the fraction of instances that have both X and Y. Well studied techniques for mining association rules exist [7]. At the heart of the mining process is the calculation of the conditional probability $P(Y|X)$ which is performed by simply counting how many of the instances that contain X also contain Y.

In the context of configuration, an association rule captures the fact that if a user selects a particular combination of options for a group of variables, she is likely to also select a certain configuration of options for a different group of variables. This knowledge comes from a data set of configurations of individual users, which we will call the *crowd data set*. In many systems, such as on-line social networking or web-based email systems, such data sets are readily available to e.g. administrators and owners.

In our case, to ensure the validity of the rules, we assume that the crowd data set comes from a group of expert users. Then, when a new and novice user makes use of the association rule $X \longrightarrow Y$, she is actually using the left-hand side of the rule to identify with a subgroup of experts whose preference with respect to X match. Then she can use the right-hand side Y as expert advice for configuring the corresponding variables. Effectively, the result is a *recommender system* based on association rules. Although there are different technologies on which a recommender system can be based [2], association rules have been shown to be effective [17], and, in our case, allow for interactive on-the-fly elicitation of the similarity criterion. To construct such interactions we make use of MDPs as we describe below.

4. FINDING OPTIMAL DECISION SEQUENCES

4.1 Markov Decision Processes

Markov decision processes (MDP) is a mathematical framework for modeling and solving sequential decision making and optimization problems under uncertainty [3]. In MDPs the decision making problem is described by a finite or infinite set of states $s \in S$. Each state is a description of the system at a particular stage and holds all the necessary information to predict the next state. The information can be described through a set of variables, each combination of values of whom uniquely describes each state. At any state, the decision-making agent chooses an action α from a set of actions A. Performance of an action has various possible outcomes, each with a different probability. Such non-deterministic effects are described by a probability distribution: for each action α, p_{ij}^{α} is defined for every pair of states s_i and s_j as the probability of reaching state s_i from s_j by taking action α. Thus each action α comes with a *transition matrix* containing p_{ij}^{α} in its cells.

Furthermore, in MDPs each state has a value and each action comes at a cost. Functions for *reward* $R : S \times S \mapsto \mathbb{R}$ and *cost* $C : A \mapsto \mathbb{R}$ are used to state the immediate reward of reaching the new state and the costs associated with taking the action, respectively. A reward matrix can be used to represent R, and a table can be used to assign a cost to each action. The overall *utility* of the transition is the reward obtained by attaining the transition minus the cost incurred by performing the action.

The core problem of MDPs is to find the course of action, called *optimal policy* π that maximizes the cumulative expected utility of each state. The latter is, roughly, the result of progressively tallying up the product of probability and utility while traversing each sequence of actions and calculating potential transitions actuated thereby. Optimal means that there is no other policy that can give the agent a larger expected cumulative utility. MDP solvers are tools that allow calculation of optimal policies given a complete MDP formulation. MDP solvers adopt various algorithms, primarily Value Iteration (VI), and Policy Iteration (PI); details on those algorithms are beyond our scope and can be found in [3].

4.2 Building the MDP

We now present the details of modeling our problem as an MDP, focusing on each of the elements discussed above.

Variables. In our translation, each variable represents the *state of our knowledge* of what option the user wants for a particular configuration variable. Recall that a configuration problem is a set of configuration variables $V_i \in \mathcal{V}$ each drawing options o_j^i from a set $\mathcal{O}_{V_i}$. The MDP variables are exactly the same set $\mathcal{V}$ and each variable V_i has the same domain $\mathcal{O}_{V_i}$ but with one important difference: to each domain $\mathcal{O}_i$, containing options $o_1^i, o_2^i, \ldots$ we add an extra option o_0^i. This additional option, which we call the "*unknown*" value, denotes that we actually don't know what option the user prefers for the configuration variable.

Actions. Each action represents a prompt to the user to make a configuration decision, i.e., a *question*. This question is associated with a configuration variable we are interested in: we ask the user what option she prefers for that variable. We denote that action as $V_i?$, where V_i is the variable in question. In her answer, the user will respond with an option for that variable. In MDP terms, asking the user a question and obtaining an answer is an action which causes the system to transition from one state to another state, changing the value of the corresponding variable from o_0^i (the "unknown" value) to some other value, based on the user response.

Rewards and Costs. Each action, i.e., asking a question, comes with one (1) unit of cost. This cost is a direct representation of the effort it takes for a human to think about the question and make a configuration decision. Each transition, on the other hand, comes with as many units of reward as the number of variables whose preferred option becomes known. We describe below how this is calculated.

Transition Matrices. Recall now that the transition matrix for each action $V_i?$ is an $N \times N$ table, where N is the number of all possible option combinations of variables V_i, i.e., all possible MDP states. Each cell represents the probability of transitioning from one state to the other as an outcome of performance of $V_i?$. In our case, the numbers are taken from the crowd data set. We particularly measure the frequency at which a certain answer occurs, given (if applicable) all previous answers from the user. Subsequently, in order to take advantage of the association rules and find decision sequences with better utility, we manipulate the transition table as follows.

Firstly, let us consider the association rules. Of all the rules that we may have discovered during the mining phase we filter those that exceed certain high confidence c and support s threshold. Then, we perform a preprocessing step in which we merge rules that have identical left-hand side.

The subsequent steps are best described though reference to a directed graph. The set of nodes of the graph is the set of all possible configurations of $\mathcal{V}$; thus, each node is an MDP state. Edges represent MDP transitions. Initially, we add only edges to the graph that are one step transitions, i.e., transitions from states in which $n-1$ variables are known to states in which n variables are known. Each edge from state S to state S' is labeled with a reward value of 1, a cost value of 1, the probability calculated as above, and the action it corresponds to.

Subsequently, we update the edges based on association rules as follows. For each association rule, we find all pairs of states S and S' such that: (a) the left hand side of the rule satisfies both S and S', (b) each variable of the right-hand side of the rule appears (b-i) with all values exactly the same as in the rule in S' and (b-ii) with all values set to unknown in S, (c) all other variables not mentioned in the rule are set to the same values for both S and S'. For each such pair, we take all edges that are targeting S and we move them so that they now target S'. Moreover, we increase the reward of each of those edges by the difference in the number of unknown variables between S and S'. Intuitively, the rule causes S to become inaccessible: whenever a user response leads to S, thanks to the information in the rule, we can actually "shortcut" to S', ignoring S. While the cost is the same, reward increases by the number of unknowns that we discover by taking the shortcut.

The resulting graph is ready to be transformed into transition and reward matrices. These are constructed initially to be zero matrices. Then for each edge labeled with the action at hand, we set to the corresponding cell the prob-

ability or the reward label of the edge, for transition and reward matrices, respectively.

The complete algorithm can be seen in Figure 1.

INPUT: a set of configuration variables, a set of association rules
OUTPUT: A Markov Decision Process model
// Build a graph of possible transitions
// Each node represents a state of knowing the option of each variable
For each possible combination of options
    Create a node in the graph
// Pre-process rules
For each set m of assoc. rules of the form ($L \rightarrow r_1, L \rightarrow r_2, \ldots, L \rightarrow r_m$),
    Merge into one rule of the form ($L \rightarrow r_1 \ldots r_m$)
// Add initial links
For each node S
  Let N be the number of cases in the crowd data set.
  Let $c(S)$ be the num. of records in the crowd data set that satisfy the known part of S.
  Let $c(o_x^a)$ be the num. of records in the crowd data s.t. $V_a = o_x^a$
  For each variable V_a such that $V_a = o_0^a$ (is *unknown*) in S
    For each possible value o_x^a of V_a
      If $o_x^a == o_0^a$ then //i.e., circular link
        Add an edge e from S to S
        $e.reward = -1, e.probability = 1, e.action = V_a?$
      else
        For each node S' that is same as S except for $V_a = o_x^a$
        Add an edge e from S to S'
        $e.reward = 0$
        If ($c(S) > 0$) then $e.probability = c(S')/c(S)$
            else $e.probability = c(o_x^a) / N$
        $e.action = V_a?$
//Update the graph based on association rules
For each association rule ($l_1 \ldots l_n \rightarrow r_1 \ldots r_m$), l_i, r_i of the form $V_i = o_.^i$
  For each pair of states S and S' such that:
    (a) variables of $l_1 \ldots l_n$ have known and same options in S and S'
    (b) variables of $r_1 \ldots r_m$ are unknown in state S and known in S'
    (c) variables not mentioned have same values in both S and S'
    For each incoming edge e to S
        Change the destination from S to S'
        $e.reward = e.reward + 1$
// Transform graph into MDP components
For each action $V_a? \in A$
    Create an empty transition matrix for $V_a?$, $T_{V_a?}$
    Create an empty reward matrix for $V_a?$, $R_{V_a?}$
    For each edge e in the graph from S_i to S_j s.t. $e.action = V_a?$
        $T_{V_a?}[i,j] = e.probability$
        $R_{V_a?}[i,j] = e.reward$

Figure 1: MDP Construction Algorithm

Solving the MDP. Having constructed the MDP model, an MDP solver can be used to calculate the optimal policy. Such policy associates each state of knowing of the so far acquired (or inferred) configuration values, with an action that represents the next configuration question to be asked. An adaptive configuration dialog can, thus, directly incorporate the resulting policy in order to decide what the next configuration question should be based on previous answers it has received from the user.

5. EVALUATION

In order to evaluate our technique we performed two studies. The first one is aimed at evaluating the applicability of the method in a real world system and assessing the degree by which the approach really saves configuration effort. The second study aims at understanding the size of configuration domains our approach can pragmatically cope with. We present these two studies in the following subsections.

5.1 Case Study: Facebook

5.1.1 Goals and Method

We applied our technique to the privacy configuration of Facebook. Facebook[1] is one of the most popular social networking systems today with more than 1.28 billion active users (April 2014). In Facebook, like in other social media, a user can avail in her personal profile area a variety of material (photos, text, information etc.) to be viewed and commented upon by other users who have been given access to them. Given that the personal material on Facebook is often of sensitive nature, how each user can control the access levels of others to her data is crucial.

The system offers a set of configuration variables that the users can configure in order to meet their privacy needs. These variables are configured using the traditional approach: a set of default options is set upon creation of a new account, and users can adjust them by going through a set of configuration screens, where the variables are presented together with possible options; the latter displayed in form of listboxes or other menus.

In our study, we isolate a set of nine (9) Facebook configuration variables. As of April 2014 (Facebook configuration screens change frequently) the configuration items for this study were located under the following three sections: General Privacy, Timeline & Tagging, and Friend List Related. The domain of settings (i.e., the set of possible values) for theses configurations vary from static options to customized dynamic lists.

We acquire a data set containing the configurations of these variables as they have been set by forty-five (45) users. The users are undergraduate students of the School of Information Technology, York University, attending the last author's class on Human Computer Interaction; the users are given bonus marks for participation. To collect the data, the users simply take screenshots of the configuration screens that contain the variables in question with their present configuration and send them over to the researchers – after erasing any personal information that may exist in those screenshots. The result is a data set amenable to association rule mining and complete application of our framework.

5.1.2 Results

The questions we aim at answering in this study are: (a) do any association rules of substantial significance emerge? (b) how much configuration effort on average does our technique save? (c) what is the average accuracy of the automated inference that our technique implies?

With regards to question (a), we mined the association rules in our data using Weka[2]. We used different thresholds for significance as well as support. By setting significance threshold to 90% and support to 10%, we acquire 738 association rules. If we set maximum significance (100%) then Weka outputs as many as 675 association rules. If we increase the support level to 15% we can still retrieve 165

[1]http://www.facebook.com
[2]http://www.cs.waikato.ac.nz/ml/weka/index.html

rules. The above means that association rules exist to a significant extend in our data set.

With respect to configuration effort – question (b) above – if each of the users had followed our technique to configure their system they would have taken on average 6.6 steps (max 9, min 4); compared to 9 that it takes to configure without assistance. This means that at least two configuration options on average (27.7%) would be decided automatically. Although, in general, effort savings is bound to depend on the amount and quality of correlations within the data set, we find that our result constitutes good evidence of the practicality of our technique.

In terms of evaluation question (c), i.e., the accuracy of configuration predictions, we conducted the experiment with confidence and support thresholds 100% and 15%, respectively. The 9-fold cross validation shows a precision of 75% in the recommended items, calculated as explained above. The fact that this precision score is away from values such as 50% or lower, tells us that the configuration agent performs sensible decisions based on the data and not e.g. randomly/arbitrarily. This offers us some confidence of the sanity of the approach.

5.2 Performance

In the second study we investigate how the performance of the MDP solver varies as the number of configuration items and hence state space increases. In the absence of an actual configuration data set of sufficient size for our performance experiments, and without visible loss of generalizability, we chose to use a publicly available benchmark data-set for association rules. Thus, we used UCI KDD's archived Mushrooms dataset[3], which has been shown to generate long patterns with high confidence. In its full size, the Mushrooms dataset is a multivariate dataset that contains 8416 instances with 23 categorical attributes and domain size ranging from 2 to 12 categorical values. In the context of software configuration we use this data set as if it were a complete configuration data set: we assume each attribute to be one configuration variable and each value a configuration option. Consecutively, each instance hypothetically stands for the configuration values chosen by a user. The MDP solver we chose is the "Markov Decision Processes (MDP) Toolbox" offered by MathWorks Inc[4]. We run the MDP toolbox in Matlab R2012b. The input to the MDP solver is sparse matrices that are generated by our Java program and are based on the extracted association rules from Weka. The experiment was conducted on a Core(TM) i5 CPU M450 2.40 GHz with 4.00 GB RAM under Windows 7.

To test the performance of the MDP solver for various numbers of configuration items, we randomly choose a subset of varying numbers of attributes in the dataset. For each subset of attributes, we extract the top 500 (wrt. support) association rules with confidence 100%; support was ranging from 36% to 11%. In Figure 2, the time to solve the policy is given with respect to the number of states, expressed as the number of equivalent binary configuration variables. Specifically, N binary variables generate a space of 3^N states, and, as such, if M are the states, the equivalent binary variables are $log_3(M)$. Note that the base is 3 instead of 2, because we add one more value (the "unknown" value) in the domain.

[3]https://archive.ics.uci.edu/ml/datasets/Mushroom
[4]http://www.mathworks.com/matlabcentral/fileexchange/25786-markov-decision-processes–mdp–toolbox

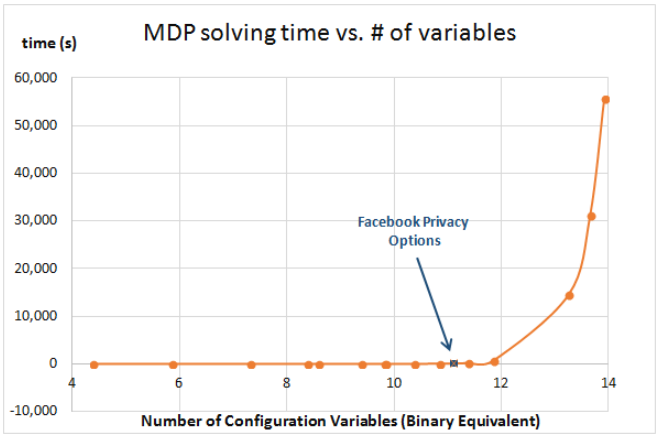

Figure 2: MDP Performance vs. Problem Size

6. RELATED WORK

The problem of configuring existing systems has been studied from a variety of angles. Wendy E. Mackay offers one of the earliest and most influential empirical studies on the problem from a Human Computer Interaction viewpoint [14]. Among other things, she finds that one of the factors that actually trigger customization effort is observing other users achieving it. In the Internet era, 23 years later, this seems to support a vision for crowdsourcing configuration. For example, a technique for allowing users help each other in performing complex computer tasks based on capturing interactions and uploading them to a repository for reuse by non-expert users has been proposed [10]. Our proposal differs in many regards including increased automation and probabilistic reasoning.

Viewing software customization as a requirements problem has also been attempted. Use of goal models [15] has been proposed to connect high-level user goals with low-level software configurations [12]. Elsewhere, declarative approaches to configuration have been put forth [11] where preference specification [13] replaces the manipulation of configuration variables. Nevertheless, for these approaches to be applicable, both the goal models and their mapping to software variability must be pre-established by experts.

In the area of Recommender Systems (RS), conversing approaches for eliciting user preference and utility have been proposed. Boutilier et al. explore different interactive elicitation strategies for optimally applying the minimax regret criterion to elicit utility in a configuration problem [4]. Similarly, Stolze and Ströbel present a technique for finding the optimal question and answer path towards recommending a product [19]. As opposed to ours, both approaches are based on utility estimation, the latter also focusing on commodity selection rather than configuration.

Efficient decision making for configuration is also relevant in Software Product Line (SPL) Engineering where a set of variation points needs to be managed and bound to derive individual SPL members [18]. In DecisionKing for example [6], hierarchies of questions are defined and integrated with asset models to facilitate product derivation. Feature models are often the basis for interactive configuration based on automated reasoning tools, such as constraint [21] or SAT [9] solvers. Ways to optimize the order of the variability

binding process based on measures of selectivity of features have also been proposed [5, 16]. Our technique differs from those in that (a) we use crowd data to infer constraints, (b) we use probabilities to predict likely answers to questions. Work that uses probability from crowd data has been proposed but for purposes different to ours, e.g. to detect misconfigurations over an observed configuration and a set of suspected sick ones [20]; an important difference from our case is that we do not have prior classifications of configurations.

7. CONCLUDING REMARKS

As modern software systems increase in size and complexity, the process of configuring them becomes more cumbersome and difficult to organize. We presented a method for designing adaptive software configuration dialogues through utilizing configurations from a crowd of existing users. Association rule mining is used to identify configuration patterns which are then used to inform the construction of an MDP model, so that the latter computes optimal policies to be used for designing the elicitation dialogs. Our contributions include that (a) we address the problem of systematically designing adaptive configuration elicitation dialogs, (b) we minimize the average dialogue length and (c) we utilize crowd wisdom for model construction.

The main item in our future research agenda is to address the scalability limit of the current approach. Firstly, we plan to study ways to encode the configuration problem so that the current technique is applicable to larger variable sets. One possibility is to break large configuration problems into smaller ones based on configuration aspects, i.e., subsets of configuration variables that are conceptually coherent. The privacy configurations of Facebook, which are of an applicable size, offer a good example of such an aspect. Other possibilities include merging variables and making domains coarser. Secondly, we plan to investigate ways to depart from MDPs and either introduce simplifications that are less computationally hard or use formalizations for which more efficient solvers exist. For example, by formalizing the problem as a Boolean satisfiability one would allow utilization of today's very efficient SAT solvers. While outputs of such approaches would likely disregard the probabilistic aspect and would probably be insensitive to user answers, they may turn out to offer a good enough solution for larger problems.

8. REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD'93)*, 1993.

[2] X. Amatriain, A. Jaimes, N. Oliver, and J. M. Pujol. Data mining methods for recommender systems. In *Recommender Systems Handbook*, pages 39–71. Springer, 2011.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999.

[4] C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artificial Intelligence*, 170(8-9):686–713, 2006.

[5] S. Chen and M. Erwig. Optimizing the product derivation process. In *Proc. of the 15th International Software Product Line Conference (SPLC'11)*, pages 35–44, 2011.

[6] D. Dhungana, P. Grünbacher, and R. Rabiser. DecisionKing: A flexible and extensible tool for integrated variability modeling. In *Proc. of the 1st International Workshop on Variability Modelling of Software-intensive Systems (VAMoS'07)*, pages 119–128, 2007.

[7] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining—a general survey and comparison. *ACM SIGKDD Explorations Newsletter*, 2(1):58–64, 2000.

[8] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements analysis for customizable software: a goals-skills-preferences framework. In *Proc. of the 11th IEEE Requirements Engineering Conference*, pages 117–126, 2003.

[9] M. Janota, G. Botterweck, and J. Marques-Silva. On lazy and eager interactive reconfiguration. In *Proc. of the 8th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'14)*, pages 81–88, 2013.

[10] N. Kushman and D. Katabi. Enabling configuration-independent automation by non-expert users. In *Proc. of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2010)*, pages 223–236, 2010.

[11] S. Liaskos, S. M. Khan, M. Litoiu, M. D. Jungblut, V. Rogozhkin, and J. Mylopoulos. Behavioral adaptation of information systems through goal models. *Information Systems*, 37(8):767–783, 2012.

[12] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. Easterbrook. Configuring common personal software: a requirements-driven approach. In *Proc. of the 13th IEEE International Requirements Engineering Conference (RE'05)*, pages 9–18, 2005.

[13] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos. Representing and reasoning about preferences in requirements engineering. *Requirements Engineering*, 16(3):227–249, Sept. 2011.

[14] W. E. Mackay. Triggers and barriers to customizing software. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI'91)*, pages 153–160, 1991.

[15] J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu. Exploring alternatives during requirements analysis. *IEEE Software*, 18(1):92–96, Jan 2001.

[16] A. Nohrer and A. Egyed. Optimizing user guidance during decision-making. In *Proc. of the 15th International Software Product Line Conference (SPLC'11)*, pages 25–34, 2011.

[17] J. J. Sandvig, B. Mobasher, and R. Burke. Robustness of collaborative recommendation based on association rule mining. In *Proc. of the 2007 ACM Conference on Recommender Systems (RecSys'07)*, pages 105–112, 2007.

[18] K. Schmid, R. Rabiser, and P. Grünbacher. A comparison of decision modeling approaches in product lines. In *Proc. of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'11)*, pages 119–126, 2011.

[19] M. Stolze and M. Ströbel. Utility-based decision tree optimization: A framework for adaptive interviewing. In *User Modeling 2001*, volume LNCS 2109, pages 105–116. 2001.

[20] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic misconfiguration troubleshooting with PeerPressure. In *Proc. of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 245–258, 2004.

[21] J. White, B. Dougherty, D. C. Schmidt, and D. Benavides. Automated reasoning for multi-step feature model configuration problems. In *Proc. of the 13th International Software Product Line Conference (SPLC '09)*, pages 11–20, 2009.