General Approach to Automatically Generating Need-based Assistance

Jason R. Wilson JRW@NORTHWESTERN.EDU
Electrical Engineering and Computing Science, Northwestern University, Evanston, IL 60626 USA

Michael Wransky MICHAEL.WRANSKY@BARNSTORMRESEARCH.COM
Jorge Tierno JORGE.TIERNO@BARNSTORMRESEARCH.COM
Barnstorm Research, 745 Atlantic Ave, Boston, MA 02111 USA

Abstract

We present here a novel approach to automatically generate hints based on the need of the individual being assisted. We adapt a practical framework from occupational therapy to guide the hint selection so that the assistance provided in a hint matches the need for assistance. By matching the need to the assistance, we can support the autonomy of the person, allowing a person opportunity for discovery and independent accomplishment. To automatically generate a hint, we estimate the need of the person, determine which action the person should take, and then select a hint for that action that matches the need. Since our approach uses plan-based representations and does not rely on any previously collected data of sample executions, we are able to generate hints for any new problem in a domain. To demonstrate this approach, we describe two examples of the algorithms executing in different systems.

1. Introduction

Many cognitive systems are designed to fully automate some process, such as driving a car or playing Go. However, full automation by an external system is not always desirable, as it may be preferable to enable a person to freely and independently achieve a task. In this case, a cognitive system would need to provide the necessary assistance to the person such that the person is better able to achieve a task with autonomy. Motivated by a desire to support this human autonomy, we propose an approach that provides a variable amount of assistance based upon the *need* of the individual.

Our goal of enhancing human capabilities stems from two different contexts: rehabilitation and pedagogy. In a rehabilitation context, a therapist helps a person to be able to gain or regain skills, enabling a person to do things on their own, such as walking, eating, and communicating. Essentially, the goal is to make the person more autonomous, able to act freely. To support the autonomy of the individual, a therapist may provide physical, cognitive, emotional, or social assistance (Schkade & Schultz, 1992). We focus here on cognitive and social assistance – such as hints, reminders, or suggestions. That said, our approach is derived from methods that generally apply to all forms of assistance. Similarly, pedagogical contexts also strive to enhance skills, building up the cognitive skills a person needs to independently solve problems. A learner may progress while in what

Vygotsky refers to as the "Zone of Proximal Development" (Vygotsky, 1986), and an instructor may support this by providing *scaffolding* to guide the student until the student has internalized the necessary concepts (Bruner, 1985; Maybin et al., 1992).

In both of these contexts, an assistant must align with the individual's goal of being able to act independently and autonomously. An approach to provide automated assistance must respect and support the individual's autonomy. In giving assistance, a system must provide enough assistance such that the person is able to make progress but not so much assistance that the system is doing most of the work, and the person is not gaining the benefits of developing a skill.

In an effort to generate assistance that supports the autonomy of an individual, we present here a novel approach to generating automated assistance in the form of lexical hints. Our approach, adapted from a framework for assistance used in occupational therapy practice (Rogers & Holm, 1994), selects an assistance based on an estimate of how much need for assistance the person has. Furthermore, our approach has a clear advantage over data-driven approaches, as we are able to adapt to novel problems and domains without reliance on previous examples of how the problem may be solved. This makes our approach more practical because hints may be immediately available for any new problem in a defined domain. We demonstrate the ability to generate need-based assistance for novel problems in two systems from different domains.

In the next section, we briefly review some of the literature on automated assistance, both in rehabilitative and pedagogical systems. Then we describe our approach, detailing the representations and algorithms we have designed to provide automated hint generation. To demonstrate our approach, we then describe how our algorithms integrated in a system to diagnose light bulb failure and a system for investigating a phishing attack. Finally, we discuss the implications and limitations of our approach.

2. Background

Automated systems that provide hints vary in when to provide a hint and what is the nature or content of the hint. Most systems may provide a hint when the user requests help (e.g., (Jin et al., 2014; Conati & Zhao, 2004; Stamper et al., 2008)). Alternatively, the system may provide unsolicited help. This may happen when a user makes a mistake (O'Rourke et al., 2015) or ends up in an incorrect solution state (Rivers & Koedinger, 2014), when the current path does not lead to a desired outcome (Stamper et al., 2008), or when the system believes the user's knowledge is lacking (Conati & Zhao, 2004). Our approach is perhaps most similar to Conati & Zhao (2004) in that the timing of the unsolicited assistance is based on the state of the user. In contrast to Conati & Zhao (2004), we focus on how much need a person has rather than the lack of knowledge. While in the current work there is little distinction between these notions, focusing on need provides an avenue by which we may account for need arising from lack of motivation, emotional factors, or other cognitive challenges.

Having a progression of hints is a common approach in both rehabilitation and pedagogical systems. In a therapeutic context, one goal is to support the autonomy of the person by providing a progression of assistance. Examples of this have been demonstrated for assisting a person with dementia to do hand washing (Mihailidis et al., 2008), therapy for children with autism spectrum disorders, (Greczek et al., 2014), or assisting older adults with medication management (Wilson et al., 2018). In each of these examples, the system provides a variable amount of assistance. However, the approaches do not generalize, as conditions for when to provide different levels of assistance is specific to the domain.

In tutoring systems, a progression of hints may be employed to give a more specific hint each time. For example, the Hint Factory, which can provide hints to a student doing logical proofs, employs a sequence of 4 hints: indicate a goal to derive, a rule to apply, premises to be used, and a combined hint (Barnes & Stamper, 2008). Another example is a coding tutor that provides progressively more specific hints: inform the student that a variable needs to be declared, tell the student which variable needs to be declared, and then provide the statement for the declaration (Jin et al., 2014). These systems do not employ a generalized approach to defining hint specificity, as the progression of hints is specific to the domain of the problem. In the next section, we will describe a framework from occupational therapy to define different levels of assistance based on the need of the individual.

To generate contextualized hints, many tutoring systems use data-driven approaches, generating a graph of execution paths from past student data. When a student is attempting to solve a problem, the state of the problem is compared with previous executions to find a similar state, which can then be used to determine probable next steps (Hicks et al., 2014; Lazar & Bratko, 2014; Stamper et al., 2008). One limitation of this approach is that only some of the possible states are "hintable", meaning the student may not be able to get hints for previously unexplored states (Hicks et al., 2014). A greater limitation is that the data requirement prevents the system from being able to generate hints for a new problem. If a new problem were to be defined, say a new logic proof, a set of students will need to correctly work through the proof (without any hints), ideally exploring a breadth of possible paths, before other students can receive hints on the new problem. In contrast to this data-driven approach, we use a knowledge-driven approach, leveraging domain and hint definitions to provide hints on novel problems starting with the first attempt to solve a problem.

Another tutoring system that uses a model of the domain to guide the user is Annie (Thomas & Young, 2010), which uses a planner to dynamically generate game play elements that guide the user. Instead of providing a progression of hints, Annie provides something similar in 5 types on interventions: prompt, hint, demonstrate, teach, or do. It chooses an intervention based on the gap in the student knowledge.

3. Overview of Approach

To automatically generate hints, we apply an approach used in occupational therapy practice. When assisting a client, an occupational therapist, who assists a person in gaining or regaining the skills necessary for daily living, must support the autonomy of the client so that the person can freely choose how to perform the skills (American Occupational Therapy Association, 2015). In assessing how much assistance a client needs, an occupational therapist must remain aware that too much assistance can diminish a person's autonomy, creating constraints on the person's ability to freely act. Conversely, too little assistance may also have a negative impact on autonomy, as a person not able to make progress on a task may feel a reduced sense of autonomy. To guide the therapies, one may use a framework such as that defined in the Performance Assessment for Self-care Skills (PASS) Manual (Rogers & Holm, 1994), which defines 9 levels of assistance from verbal support (level 1) to total assistance (level 9). We are currently interested in verbal assistance, levels 1-3, but our approach should generalize to all 9 levels.

Adapting this to automatic hint generation, our approach revolves around matching the need for assistance to the level of assistance provided. To accomplish this, the algorithms we describe here estimate how much need a person has, determines what action an agent may take to make progress in a given task, gather the hints available for that action, and then select the hint that best matches the level of need of the person.

Our approach makes the assumption that the person has a similar goal as the agent providing assistance. In other words, the person is trying to achieve the goal that the agent is giving hints to achieve. We recognize that this may be an invalid assumption in some cases, as a person may not actually care about achieving the goal. An example of this may be a student that just wants to get to the answer and does not care about learning how to get the answer. Another example in elder care is assisting on a task like medication management when the person would rather not manage the medications and would prefer to have an automated system handle it. In both of these cases, our approach to maximizing autonomy may not hold. We feel it is safe for now to make the assumption that the person's and agent's goals align, and we plan for future work that relaxes this assumption.

3.1 Use Case: Travel

In the descriptions below, we use a travel example, a domain commonly used in demonstrating planning systems (Nau et al., 2005). In this example, a person is attempting to go from home to the park. The person may travel by foot or by taxi. If the person travels by taxi, the agent must call the taxi, then ride in the taxi, and then finally pay the driver. The hints we generate are intended to inform a person how best to get to the park, without missing any steps (e.g., paying the taxi driver).

3.2 Representation

The algorithms we describe below rely on a few representations depicted in Figure 1. To generate an assistance based on an estimated need of the person, we need to represent a hint, the level of assistance provided by the hint, an action that an agent may take, the need of a person, and the state of the problem and the user.

Hint: The goal of our work is to automatically generate a hint, where a hint is a fully grounded form of assistance that the agent can provide. Assistance is a cognitive or social aid that allows the user to continue to make progress toward a goal. Physical assistance is another valid form of assistance, but we focus on non-physical assistance at this time. The assistance is intended to be provided in a social manner, ideally spoken to the user. Currently, the only assumptions we make about how the hint will be presented to the user is that the presentation supports a lexical form of the hint. In other words, the hints generated are words that may be displayed on a screen or converted to speech. For example, a hint in the travel problem may be "You need to pay the taxi driver."

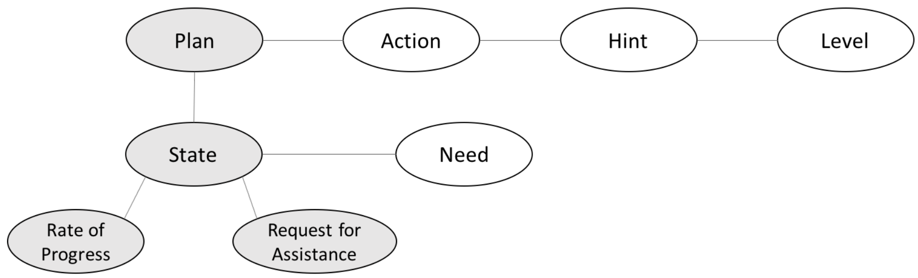

Figure 1. The hint, level of assistance, action, and need representations are key components to our approach. The shaded items (state, plan, rate, and request) are in support of the key components.

Level of Assistance: Hints are statements that the agent may make to provide assistance. Each hint is associated with a level of assistance, where the level of assistance defines how much assistance is being provided by giving the hint. In following the guidelines of the PASS manual (Rogers & Holm, 1994), we use three levels of verbal assistance. Level 1 is a *verbal support*, encouraging the user to keep going. Level 2 is a *verbal indirect*, whereby the agent gives some indication of which action to take next but does not directly specify the action. Level 3 is a *verbal direct*, where the agent directly specifies which action may be taken next to make progress towards the goal. A level 3 hint for the action `walk` is, "You could try walking to the park." This hint clearly specifies which action needs to be taken, whereas a hint like "The park is not too far from home." would be a level 2 hint, indirectly indicating that the `walk` action should be taken.

In our investigations, we have found that 3 rigid levels of assistance do not always provide the right amount of nuance. For example, there may be an assistance between level 1 and 2 that provides encouragement and support while also suggesting that there is a particular action that needs to be taken. This may occur when a user has makes an improper action and then takes a correct action. In this scenario, it may be most appropriate for the agent to give support (to recognize the correct action) while also giving an indirect hint (to assist in correcting the improper action). Since discrete levels may not be necessary, the algorithms we describe below do not assume this discreteness and instead work equally well for continuous or discrete values.

Action: Each hint is designed to guide the user to taking an action that makes progress toward the given goal. We define an action to be an operation that changes the state of the world. Actions are represented as operators in a hierarchical task network (HTN) (Nau et al., 1999). For the example problem of traveling to the park, we adapt the HTN representations defined in (Nau et al., 2005).

Each action has an associated set of hints that indicate how the user may proceed to make progress toward the goal of the task. In all the examples given here, we restrict each action to having exactly 3 hints, one for each level of assistance. However, the algorithms described below generalize to allow an arbitrary number of hints along a continuous scale of levels of assistance.

Need: The level of assistance that an agent should provide is based on how much need the person has. Since the level of assistance and the need for assistance should match, we measure need along a scale of the same dimensions as assistance. In the examples given in this paper, the scale is a continuous value that ranges from 0 to 3, with 0 indicating the user needs no assistance. The examples given here use this scale, but the algorithms described below only assume that it is a scale starting at 0 and that it does not matter what the size of the scale is or whether it is a discrete or continuous scale.

State: A representation of the state of the system encapsulates the necessary representations for the given problem solving (e.g., location of the person, how much money the person has). Additionally, the state is used to track the dynamic information that is necessary for the algorithms. This includes the plan generated at each state, the estimated amount of need, the rate of progress, and whether an explicit request for assistance was made. The plan and need are as described above. The rate of progress and request for assistance variables are described in Section 3.3.2.

3.3 Algorithms

We describe here a few algorithms used to generate a hint. The top-level algorithm outlines the overall procedure of generating the hint. Two steps in this procedure use the other algorithms described in this section.

3.3.1 Generate Hint

The *Generate Hint* algorithm (Alg. 1) outlines the procedure for generating a hint given the current state s_i of the problem. The procedure requires knowing whether help has been explicitly requested r_i in the current state and the estimated need n_{i-1} and plan p_{i-1} from the previous run of the algorithm. On the first run, the need is initialized to 0 and the plan is the sequence of actions from the initial state.

Algorithm 1 Generate Hint algorithm

procedure GEN_HINT($s_i, r_i, n_{i-1}, p_{i-1}$)
    $p_i \leftarrow plan(s_i)$ ▷ Find plan for current state
    $n_i \leftarrow estimate\_need(s_i, n_{i-1}, p_{i-1}, p_i)$
    $h_i \leftarrow find\_assist(p_i, n_i)$
    return (h_i, n_i, p_i) ▷ h_i is the hint
end procedure

The first step of the *Generate Hint* algorithm gets the plan to solve the problem from the current state s_i. An HTN planner (pyhop (Nau, 2013)) generates a sequence of operators, which are actions an agent (i.e., the human user) may execute to solve the problem. For example, the planner may return the following sequence: `[call_taxi, ride_taxi, pay_driver]`.

The second step estimates how much need for assistance the person has. The algorithm for this is described in the next section. The last step finds the appropriate hint given the plan (which indicates the next action to take) and how much need the person has. Finally, the hint (along with other variables to be used on next iteration) is returned to the system.

3.3.2 Estimate Need

Estimating the need a person has is a difficult process, as there are many factors to potentially take into account. Our approach assumes that the current need is largely based upon the previous need of the person, and other factors determine how much that need has increased or decreased. Two important factors that can be used to estimate need is whether the user has explicitly requested assistance and how much progress is being made towards the goal.

Algorithm 2 Estimate Need

procedure ESTIMATE_NEED($n_{i-1}, r_i, p_{i-1}, p_i$)
  $d_i \leftarrow length(p_i) - length(p_{i-1})$
  return $f(n_{i-1}, r_i, d_i)$
end procedure

A person may explicitly request assistance (represented in Algorithm 2 as r_i) by making verbal statements like “Help me” or “I don’t understand”. These statements need to be processed by the system before it requests that a hint be generated (Algorithm 1). Once the hint generation process begins, r_i is simply a flag indicating that preprocessing has determined that an explicit request for assistance has been made.

Progress is determined by seeing how many steps remain to accomplish the goal, where the plan is used to know how many steps remain. If the number of steps has decreased, then progress is being made and the person may have less need. Conversely, if progress has not been made, assessed by an increase or a consistent number of steps in the plan, then the person may have an increased need for assistance. Other factors that may influence need are discussed later in the context of future work.

The function $f(n_{i-1}, r_i, d_i)$ updates the need based on the previous need n_{i-1}, whether there was an explicit request for assistance r_i, and progress (or lack thereof) that has been made d_i. The function to calculate the updated need is the following:

$$f(n_{i-1}, r_i, d_i) = \min(MN, \max(0, n_{i-1} + 2 * d_i + 1 + r_i)) \tag{1}$$

This function outputs a value between 0 (the minimum need) and MN (the maximum need). To calculate this value, it starts with the previous estimated need. If a person has taken an action that moves one step towards the goal, d_i would be -1, and doubling this value makes it -2. To not decrement the level of need too quickly, 1 is added. Conversely, if the person has taken a step back, distance to the goal increases, making the need jump up by 3. Lastly, if no progress has been made (the user has taken an action that neither works toward or away from a goal), then the need increases by 1. This happens because it assumes that the person could be stuck or unsure how to proceed, indicating a small increase in need. If there is an explicit request, r_i is 1, otherwise it is 0. This makes the level of need to increase by 1 if assistance has been requested.

3.3.3 Find Assistance

Determining the appropriate assistance requires finding an action that will make progress towards solving the problem and then choosing an assistance corresponding to that action that best matches the estimated need. Algorithm 3 finds an assistance for the first action in the plan.

Algorithm 3 Find Assistance

```
procedure FIND_ASSIST(p_i, n_i)
    a ← first(p_i)
    H ← lookup_hints(a)
    t ← arg min_{h∈H} |level(h) − n_i|
    return ground_hint(t, a)
end procedure
```

The algorithm starts by getting the first action in the plan p_i and then looking up all the hint templates associated to that action. Then it selects the template that minimizes the difference between the level of need n_i and the level of assistance $level(h)$ provided with a hint. Finally, it returns a grounded hint, which replaces variables in the hint template with grounded values in the action. For example, if the first action is (call_taxi 'me' 'home') and the hint template is "Maybe take a taxi from {1}", then the hint that gets returned is "Maybe take a taxi from home."

4. Demonstrations

In this section, we present two demonstrations of how the representations and algorithms may be used to integrate into a given system and provide hints to a user. Both examples use a chat-based interface to interact with the user. The first example is relatively simple and is intended to demonstrate how estimates of need influence which hints are generated. In the latter example, the system is more complex, and we intend to demonstrate that our approach may integrate with a larger and more complex system.

4.1 Lightbulb Diagnostics

There are many tools to help a person diagnose a problem, whether it is diagnosing why the Internet is not working or why a photocopier is not working. In some cases, the purpose of the tool may not be to automatically do the diagnostics but train the person in how to diagnose similar problems. The hints should be informative but allow the person to independently discover the solution when possible. We use a simple problem of a light not working as a demonstration of a basic diagnostics problem, analogous to more complex diagnostics problems.

Interacting with the diagnostics training tool is via a web-based chat interface, where the user interacts with a computer agent that reports the state of the problem and may give hints on how to proceed. At the beginning of the interaction, the agent presents the problem to the user, and then the user may try to resolve the problem by typing in things to do. The interface uses a limited natural language interface instead of a strict set of commands, allowing the user to interact with the agent in a more natural way.

When a user enters some text, the text is passed to the interaction manager to interpret what is being said. It attempts to match what the person has entered to a valid operation in the domain. Alternatively, it may recognize that the person is explicitly requesting assistance (e.g., "help" or "I don't understand"). If the entered text maps to a domain operator, then that domain operator is

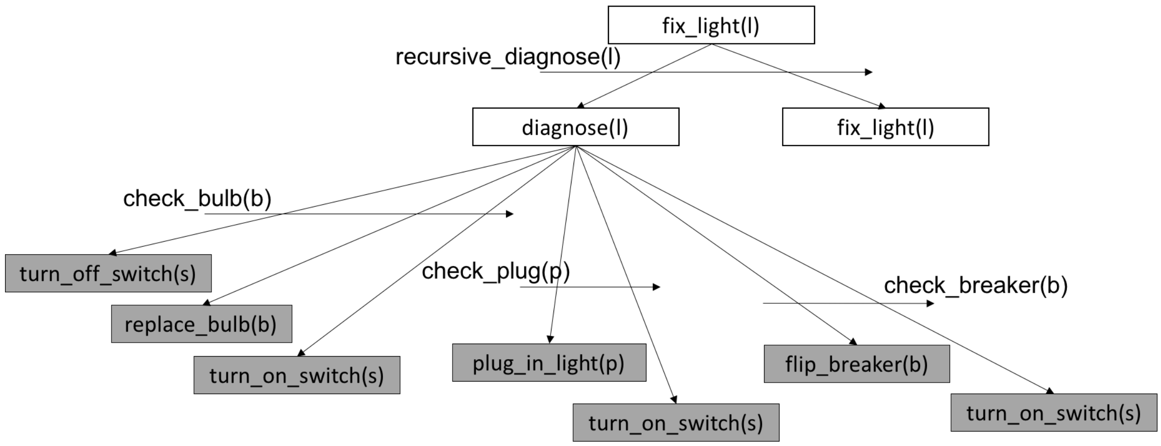

Figure 2. The decomposition tree for the task `fix_light` has tasks in the unshaded boxes, operators in the shaded boxes, and the left-to-right arrows indicate the ordering for a method. The methods `check_bulb`, `check_plug`, and `check_breaker` are different ways to accomplish the `diagnose` task. A recursive method to fix light is used to handle the case where there are reasons why the light is not working.

executed. The resulting state of the system is passed to the hint generation algorithm, which returns an appropriate hint for the current state (of the domain and of the user). The state of the domain is then translated into natural text (using frames and templates). Finally, the hint and the text are returned to the UI to be presented to the user.

4.1.1 Planning Definition

In this example, there are three possibilities for what could be the source of the problem: bulb is bad, the light is unplugged, and/or the power is out. The HTN definition shown in Figure 2 has a method for each these approaches. Each method grounds out in operators, consisting of turning a switch on/off, replacing a bulb, plugging a light in, and flipping a breaker.

4.1.2 Hinting Definition

Each operator has an ordered list of three possible hints, see Table 1. The hints represent assistance for levels 1-3. The level 1 assistance is a *verbal support*, intended to encourage the user to keep making progress. The level 2 assistance is a *verbal indirect*, designed to give some indication as to how to proceed without directly specifying which action. Lastly, the level 3 assistance is a *verbal direct*, which gives a strong indication as to which action the user should take next.

4.1.3 Example Discussion

In this example shown in Figure 3, the user solves a diagnostics problem to fix a situation where a light does not work. The natural first step would be to turn on the light. When that does not work, the user has some options and chooses to try the most obvious solution: turn on the light switch. Since the switch is already on, turning on the switch has no effect, and the system responds with the

Table 1. Each operator has a sequence of three possible hints.

Operator	Level 1	Level 2	Level 3
turn_off_switch	Ok	Check the switch	Turn the light off before proceeding.
turn_on_switch	Almost there	Check the light switch	Turn the switch on to see if the light works now
replace_bulb	What should you check next?	Is the bulb good?	Maybe try replacing the light bulb
plug_in_light	Good	Is the light plugged in?	Try plugging in the light
flip_breaker	Nice job	Is the power on?	Try flipping the breaker

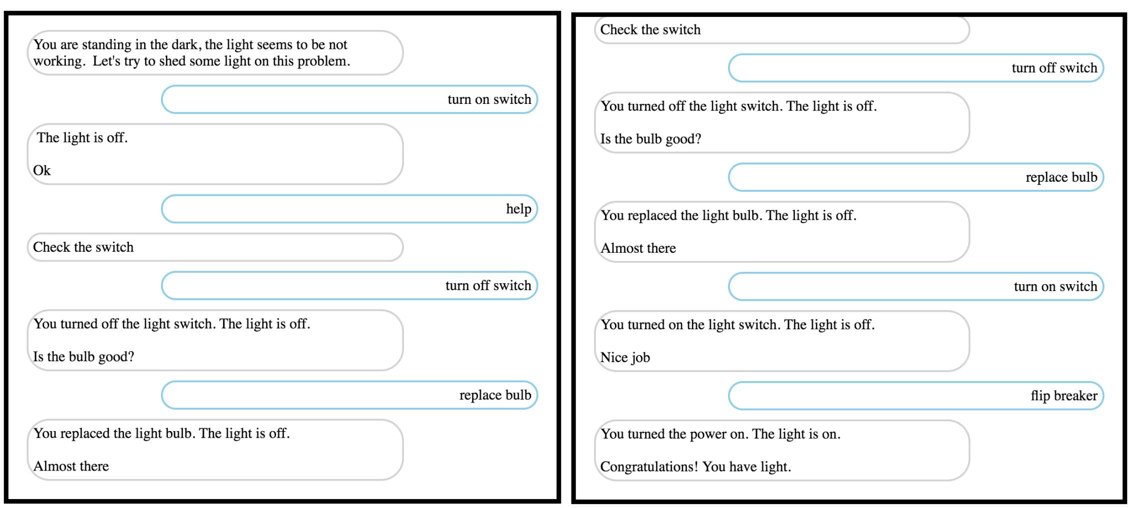

Figure 3. The user gets some assistance with diagnosing why the light does not work. The text on the left is provided by the system, and the text on the right is provided by the user.

unchanged state and a level 1 assistance, “Ok”. In this case, even though the user has taken an action that does not make progress, but the estimated need is still low. As a result, the system provides a level 1 assistance and not a level 2 assistance yet. The level 1 assistance in meant to encourage the user to keep going and try to figure it out independently.

When the user then explicitly makes a request for assistance, the user is provided with a small bit of assistance, “Check the switch”. When the user turns off the switch, the system reports the updated state. Since the level of need may not have gone down too much yet, the system continues to provide a level 2 assistance, “Is the bulb good?”.

The user then has the bulb replaced, and the system reports the updated state. At this point, the level of assistance has gone back down to a level 1 and gives a small encouragement, “Almost there”. Finally, the user can turn on the switch and flip the breaker to get the light back on.

This example shows multiple levels of assistance, solicited and unsolicited assistance, and a dynamic need estimate that does not reset at each new state. Furthermore, this example is just one of 8 possible problems that can be defined for this domain, and no additional effort is necessary to supply hints for all variants.

4.2 Phishing

Human computer interfaces are becoming increasingly prevalent to streamline supervisory tasks of complex systems, such as a power grid. Even with these interfaces, supervisors are inundated with alarms, sometimes inaccurate, which lower the supervisor’s ability to maintain a concise understanding of the current system state to maintain safety and security of all systems. Other times, no alerts are produced, which may decrease the supervisor’s ability to quickly resolve an urgent alarm. Overall, users of human computer interfaces face challenges related to situational awareness of the system.

In order to assist the user to maintain appropriate situational awareness, tactical decision games (TDG’s) are used to engage the user and promote a continual stream of problem solving. We demonstrate the role of hints in such a TDG in the realm of a human computer interface utilized by a supervisor of a power grid. Power grid supervision incorporates both physical systems (transformers, generators, etc.) as well as cyber systems (networks, firmware, servers, etc.). For the purposes of simplicity, we develop a TDG to assist the user in maintaining situational awareness of a potential phishing attack. Phishing attacks are characterized by a malicious actor sending emails to employees of the power grid either seeking to trick the employee to download malware or send along login credentials. Depending on the sophistication of the malicious actor, phishing attacks range in complexity, and a supervisor may need to investigate multiple separate routes before discovering the appropriate method to prevent the attack to continue.

The TDG is presented to the user as part of the interface via a chat interface. Messages are passed between the TDG and user, where the user specifies actions to take, the game executes the actions and presents the updated state. User inputs are feed through a NLP model that maps phrases to either game actions understood by the system or a hint request. The NLP model utilizes word and sentence embeddings in a learned recurrent neural network sequence-to-sequence architecture, making it capable of processing regular human readable text. Crafting a model capable of inter-

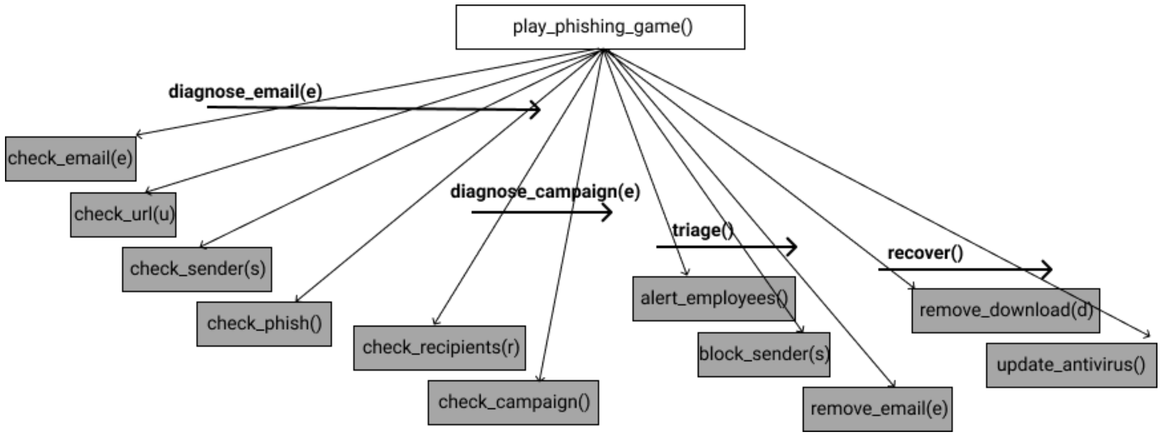

Figure 4. The decomposition tree for the task `play_phishing_game` has tasks in the unshaded boxes, operators in the shaded boxes, and the left-to-right arrows indicate the ordering for a method. The methods `diagnose_email`, `diagnose_campaign`, `triage`, and `recover` represent the steps a user must accomplish in completing the game. In this scenario methods must be followed in the proper order from left to right, otherwise the user would have insufficient information needed to complete the next method.

preting variable inputs and mapping them to the same operator allows for the user to require less training in playing the TDG and encourages better gameplay.

4.2.1 Planning Definition

Within this domain of cyber security, experts rely on what are known as "playbooks" to determine appropriate counter measures to threats. Playbooks often adopt decision tree like structures in which the result of step 1 may lead to steps 2-4, or may allow the user to jump to step 5. Because of this, playbooks provide the initial structure for the planner to model. For phishing attacks, the playbook is a simple linear sequence of steps, where information from prior steps help inform the decision for the current step. Modeling the planner in a similar linear progression makes the TDG play like a mock drill, reinforcing the protocols supervisors need to use, and simultaneously preparing them in case a phishing attack occurs.

Following the architecture suggested by a HTN solver, playbook steps are constructed as operators. These operators may or may not live under the umbrella of a more general method, but for this phishing example each playbook step is an operator with no generalized method combining any of them. See Figure 4 for a visual flow of the operators. The phishing TDG maintains a simple state in which each operator is a field in the state with a True/False flag to indicate if it has been completed. Each operator contains logic to determine if the game state contains the appropriate values to permit the operator to run (so step 3 cannot run before steps 1 and 2 are marked complete). When solving for a plan, the output is the sequence of operators that correctly take the game from start to finish.

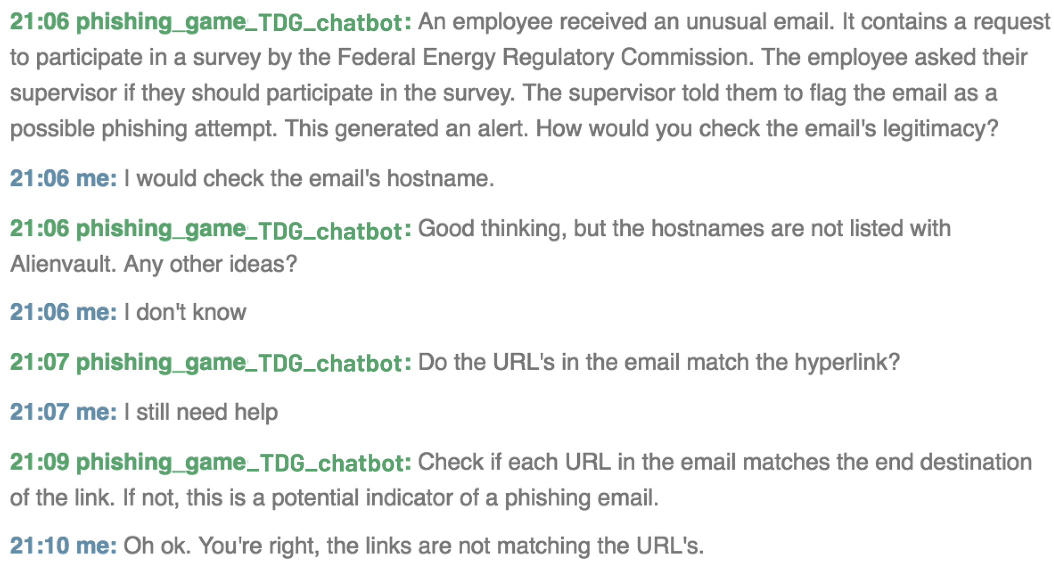

Figure 5. Actual screenshot of chatbot and hinting algorithm in use. As the user progresses through a tactical decision game based on a phishing campaign, the hinting system aids the user recall appropriate steps to conduct in diagnosing the situation.

4.2.2 Hinting Definition

For each operator, three hints of varying specificity are crafted, see Table 2. These hints range from general clues for users who have a low need, to specific suggested inputs for users with the highest level of need. Hints are provided to the user only when requested, for example when the user types the message "I need help." When a user requests for a hint, the planner is executed, creating a plan taking the user from the current game state to the goal state. The first operator from this solution path is extracted and used to determine which list of hints to pull from.

4.2.3 Example Discussion

In this example, shown in Figure 5, the user plays an example decision game focused on evaluating a potential phishing campaign. The game begins with the chatbot presenting the user with a scenario: a suspicious email has been flagged and the game user needs to evaluate the email. In accordance to standard phishing diagnosis playbooks, the user instructs the chatbot to inspect the email's hostname, and the game informs the user that if it is a phishing email it has not been marked as such by others. At this point, the user becomes stumped and tells the chatbot "I don't know". The chatbot interprets this as a request for help and the hinting algorithm is used to produce a level 1 hint, which is designed to get the user thinking of the next step. Despite this hint, the user still needs additional assistance and requests another hint. The chatbot returns with a level 2 hint that provides clearer directions. This level of assistance meets the user's need and allows them to recall the next procedure in the playbook.

Table 2. Each operator has a sequence of three possible hints.

Operator	Level 1	Level 2	Level 3
check_email	Good thinking	How can you tell if any hostnames are a known threat?	Check hostnames of all links in the email.
check_url	Do the URL's match the hyperlink?	Check if each URL in the email matches the end destination of the link. If not, this is a potential indicator of a phishing email.	Indicate that the URL's and hyperlinks do not match.
check_sender	Keep going to try something else	Have you checked out the sender's email address?	Verify sender's domain name.
check_phish	Ok what does this information tell you?	Do you think there is something suspicious about this email?	Specify if email is a phishing attempt.
check_recipients	Good work! What next?	Did other employees receive email from the same sender?	Check number of recipients.
check_campaign	You are right, a lot of employees got this email	Does this indicate a larger campaign?	Classify this email as part of a larger campaign or not.
alert_employees	Now that we have determined this is a phishing campaign, who do you tell?	Do the employees know about the attempted hack?	Send out a mass alert to all employees
block_sender	Nice job, but there is still more work to do.	How can we prevent future emails from this address?	Blacklist the sender.
remove_email	The address is now blocked, but more needs done	Can you prevent employees form opening email they have already received?	Remove the email from all employee inboxes
remove_download	Good work! Is there anything you can do to protect the network?	Can you prevent the malware from downloading even if an employee clicks the link?	Blacklist all links in email to prevent downloads.
update_antivirus	Almost done! Is there a way to improve the system?	Did your antivirus software detect this threat?	Update antivirus software.

Through this game example, we present how playbooks can be transformed to a game setting. Such games are useful not only in refreshing a user's memory of the appropriate protocols and courses of action, it also can enhance the user's situational awareness. Even if the next real alert given to the user to evaluate is not a phishing attempt, the user is resuming work in a problem solving mindset, instead of engaging the situation from a less prepared mentality.

This demonstrates how solicited assistance does not solve the problem for the user, but rather allows the user to recall procedures needed to progress a decision making process. Additionally, the gameplay is designed to simultaneously incorporate preparedness and training in the same setting, with the addition of hinting to enhance user learning

5. Discussion

We presented here an approach to automatically generate hints that vary based upon the need of the individual, with the goal of maximizing human autonomy. We are able to handle novel problems and easily adapt to new domains by integrating a planner into our solution. This is in sharp contrast to most other solutions that take a data-driven approach and require example executions of the problem solving in order to provide hints. By using a planner, we are able to generate hints for any problem in a domain that has been defined, as we demonstrated in our example of diagnosing the lightbulb not working. Adapting to a new domain requires a similar effort as defining a new domain for a planner, with hint generation requiring the planning domain definition plus definitions of hints for each operator. We believe this more general approach makes our solution applicable to a broader range of applications, from rehabilitation and therapy to training and tutoring.

Having based our work on an established and regularly applied framework from occupational therapy, we have reason to believe that our approach achieves our goal of supporting autonomy. However, work remains to validate the approach, and this includes conducting evaluations with users to assess how well their autonomy is supported. Additionally, we need to validate mechanics within our approach, namely, the need estimation and the accuracy of the levels of assistance. Ongoing work is designing a user evaluation of the approach to verify that the user's autonomy is supported while being assisted in a problem-solving task.

We are also investigating improvements to inferring whether a person has a need for assistance. Our current estimation of need is based on progress in the task, explicit request for assistance, and the previous level of need of the individual. We plan to incorporate other factors that may allow for a more accurate estimation of need, such as facial expressions (D'Mello & Graesser, 2010; Ghali et al., 2016), lexical tone (D'Mello & Graesser, 2010), and gaze (Ghali et al., 2016; Wilson et al., 2018). As more dimensions are included in the estimate, it will be paramount that we devise a means for validating our need estimates.

With our approach based on matching need to assistance, in addition to validating our need estimates, we also need to verify the accuracy of the levels of assistance in the hints that are generated. In particular, we will verify that a level n assistance is more assistive than a level $n-1$. We are confident that our hints follow this constraint because we have authored the hints under the framework defined in the PASS Manual (Rogers & Holm, 1994). In other words, our level 1 hints provide indirect support, and our level 2 hints directly indicate what action the user should take. However, empirical evidence will be needed to verify that our level 2 hints are indeed more assistive than our level 1.

While our model-based approach using a planner allows us to readily adapt to novel problems, there are two main limitations to this approach. First, defining a new domain for a planner may incur a notable cost to develop the domain model (McCluskey et al., 2017). In the examples we have demonstrated, this cost was partially mitigated by the fact that plan operators served a dual purpose. In addition to being used by the planner, the target systems (lightbulb diagnostics and phishing investigation) used the same plan operators as user actions in the system. Second, how we integrate a planner is that our approaches requires a planner to be invoked after each action. While this provides the flexibility to adapt to the user, it could be too costly to regularly invoke a planner. To mitigate this cost, we will leverage previous planning outcomes, perhaps using analogical inferences to identify possible actions to re-enter a plan (Friedman et al., 2017). As a result, we will be able to improve performance as more data becomes available while maintaining our ability to operate on novel problems.

6. Conclusion

We have presented a novel approach to automatically generating hints based on the need of the individual being assisted. By adapting a practical framework from occupational therapy to inform how to assist a person while supporting the autonomy of the individual, our approach may be broadly applied to both rehabilitation and pedagogical contexts. By integrating a planner, we are able to provide hints in novel problems and domains. Within a given domain, our approach is able to provide hints to novel problems on the first encounter and is not reliant on any previous executions or solutions to the problem. We have demonstrated this approach by integrating the algorithms into 2 separate systems.

Acknowledgements

The research in this paper was supported in part by DARPA, under contract D17PC00093. Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Thanks to Seongsik Kim for his assistance with the lightbulb diagnostics system.

References

American Occupational Therapy Association (2015). *Reference guide to the occupational therapy code of ethics (2015 edition)*. Technical Report 6913410030, American Occupational Therapy Association, Bethesda, MD.

Barnes, T., & Stamper, J. (2008). Toward automatic hint generation for logic proof tutoring using historical student data. *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 373–382). Berlin: Springer.

Bruner, J. (1985). Vygotsky: A historical and conceptual perspective. *Culture, communication, and cognition: Vygotskian perspectives*, (pp. 21–34).

Conati, C., & Zhao, X. (2004). Building and evaluating an intelligent pedagogical agent to improve the effectiveness of an educational game. *Proceedings of the Ninth International Conference on Intelligent User Interfaces* (pp. 6–13).

D'Mello, S. K., & Graesser, A. (2010). Multimodal semi-automated affect detection from conversational cues, gross body language, and facial features. *User Modeling and User-Adapted Interaction*, *20*, 147–187.

Friedman, S., Burstein, M., Rye, J., & Kuter, U. (2017). Analogical localization: Flexible plan execution in open worlds. *Proceedings of the ICCBR 2017 Workshops*. Trondheim, Norway.

Ghali, R., Frasson, C., & Ouellet, S. (2016). Towards real time detection of learners' need of help in serious games. *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference* (pp. 154–157). Key Largo, FL.

Greczek, J., Kaszubski, E., Atrash, A., & Mataric, M. (2014). Graded cueing feedback in robot-mediated imitation practice for children with autism spectrum disorders. *The Twenty-Third IEEE International Symposium on Robot and Human Interactive Communication* (pp. 561–566).

Hicks, A., Peddycord, B., & Barnes, T. (2014). Building games to learn from their players: Generating hints in a serious game. *Proceedings of Intelligent Tutoring Systems: Twelfth International Conference* (pp. 312–317). Cham: Springer International.

Jin, W., Corbett, A., Lloyd, W., Baumstark, L., & Rolka, C. (2014). Evaluation of guided-planning and assisted-coding with task relevant dynamic hinting. *International Conference on Intelligent Tutoring Systems* (pp. 318–328). Springer.

Lazar, T., & Bratko, I. (2014). Data-driven program synthesis for hint generation in programming tutors. *International Conference on Intelligent Tutoring Systems* (pp. 306–311). Springer.

Maybin, J., Mercer, N., & Stierer, B. (1992). 'Scaffolding': Learning in the classroom. *Thinking Voices: The Work of the National Oracy Project* (pp. 186–195). Hodder & Stoughton.

McCluskey, T. L., Vaquero, T. S., & Vallati, M. (2017). Engineering knowledge for automated planning: Towards a notion of quality. *Proceedings of the Knowledge Capture Conference* (p. 14). ACM.

Mihailidis, A., Boger, J. N., Craig, T., & Hoey, J. (2008). The COACH prompting system to assist older adults with dementia through handwashing: An efficacy study. *BMC Geriatrics*, *8*, 28.

Nau, D. (2013). Game applications of HTN planning with state variables. *Planning in Games: Papers from the 2013 ICAPS Workshop*.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Wu, D., Yaman, F., Munoz-Avila, H., & Murdock, J. W. (2005). Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, *20*, 34–41.

Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 968–973). Morgan Kaufmann.

O'Rourke, E., Andersen, E., Gulwani, S., & Popovi, Z. (2015). A framework for automatically generating interactive instructional scaffolding. *Proceedings of the Thirty-Third Annual ACM Conference on Human Factors in Computing Systems* (pp. 1545–1554). Seoul, Republic of Korea:

ACM.

Rivers, K., & Koedinger, K. R. (2014). Automating hint generation with solution space path construction. *International Conference on Intelligent Tutoring Systems* (pp. 329–339). Springer.

Rogers, J. C., & Holm, M. B. (1994). Performance assessment of self-care skills (PASS-Home) Version 3.1. Unpublished assessment tool. Pittsburgh, PA: University of Pittsburgh.

Schkade, J. K., & Schultz, S. (1992). Occupational adaptation: Toward a holistic approach for contemporary practice. *American Journal of Occupational Therapy*, *46*, 829–837.

Stamper, J., Barnes, T., Lehmann, L., & Croy, M. (2008). The hint factory: Automatic generation of contextualized help for existing computer aided instruction. *Proceedings of the Ninth International Conference on Intelligent Tutoring Systems Young Researchers Track* (pp. 71–78).

Thomas, J. M., & Young, R. M. (2010). Annie: Automated generation of adaptive learner guidance for fun serious games. *IEEE Transactions on Learning Technologies*, *3*, 329–343.

Vygotsky, L. S. (1986). *Thought and language*. MIT Press.

Wilson, J. R., Lee, N. Y., Saechao, A., Tickle-Degnen, L., & Scheutz, M. (2018). Supporting human autonomy in a robot-assisted medication sorting task. *International Journal of Social Robotics*, (pp. 1–21).