Handling Planning Failures with Virtual Actions

Jicheng Fu, Sijie Tian
Computer Science Department
University of Central Oklahoma
Edmond, OK, USA
jfu@uco.edu, stain@uco.edu

Vincent Ng, Farokh B. Bastani, and I-Ling Yen
Computer Science Department
University of Texas at Dallas
Richardson, TX, USA
vince@hlt.utdallas.edu, {bastani, ilyen}@utdallas.edu

***Abstract*— Artificial intelligence (AI) planners have been widely used in many fields, such as intelligent agents, autonomous robots, web service compositions, etc. However, existing AI planners share a common problem: When given a problem to solve, they either return a solution if one exists or report that no solution is found. However, simply reporting failure leaves no clues for people to trace the causes of the planning failure. In this paper, we present a novel approach that can propose virtual actions in the event of planning failure. Virtual actions enable traditional planners to succeed and hence return an incomplete plan instead of merely an error message. More importantly, the specifications of the virtual actions suggest what the missing parts may contain, thus providing important clues to users as to the nature of the failure. Experimental results show that our approach constantly returns useful and comprehensible information for humans, thus making AI planning more practical when solving real-world problems.**

Keywords-Artificial intelligence planning; Graphplan; Level off; Genetic algorithm

I. INTRODUCTION

Artificial Intelligence (AI) planners, which seek to generate a plan of actions that lead us from an initial state to a goal condition, have been widely used in software engineering fields to help automate various software development tasks, such as test case generation [1], program synthesis [2], automated web service composition [3], etc. AI planning is declarative and goal-oriented, which enable users to focus on high-level specifications (i.e., what to do) and let AI planner automatically determine the low-level details (i.e., how to get things done).

Although AI planning strives to solve real world problems, its practicality depends heavily on the completeness of the planning domains. However, in the real world development processes, it is frequently noticed that not all planning domains are complete, i.e., actions in the planning domains may not be readily available for the desired system goal. For example, web services are usually modeled as planning actions and hence AI planning can be applied to web service composition [4]. It is unrealistic, however, to assume that all the necessary web services are available in the Internet. Such an incomplete domain tends to result in planning failures. In case that some web services are missing, the planner will simply return an error message of failing to find a plan. Since the planning failure does not provide any useful information, the common practice is to manually identify the usable as well as missing services. The missing ones will be developed from scratch or adapted from some existing services.

To our knowledge, when given a problem to solve, existing AI planners either return a plan if one exists or report that no solution is found. In the latter case, all the efforts in the planning process are wasted. For example, consider the popular travel reservation case study system [5]. Many actions (i.e., services), such as booking airline, hotel, shuttle, processing credit card, etc., can be composed to complete the desired trip. However, there is no guarantee that all of these services are available on the Internet. If some service is missing (e.g., the shuttle service can only be found through the traditional yellow page book), the planner will fail and no information will be provided regarding what is missing. However, if the planner is "intelligent" enough to generate an *incomplete* plan suggesting what the missing actions might be, the user may still be able to obtain useful information and can proceed to obtain the missing services via alternative means.

In this paper, we propose a novel approach to the task of generating incomplete plans when traditional planning processes fail. Two challenges need to be addressed. First, due to the missing information, a planner can neither reach the goal from the initial state nor vice versa. In particular, when employing bi-directional planning to generate plans from both ends, the two planning processes will never meet in the middle. We address this challenge by finding the gap for the bi-directional planning and proposing a *virtual action* to bridge this gap so that an incomplete plan can be returned. However, proposing the virtual action is a challenging task. The reason is that a large amount of information is usually present in the bi-directional planning process and this makes it impossible to enumerate all the possible virtual actions and identify the best one. We address this second challenge by using a genetic algorithm (GA) approach to mine useful information.

As discussed above, an incomplete plan includes two types of actions: *real actions*, which are actions defined in the planning domain; and a *virtual action*, which does not

exist in the domain but is treated in the same way as real actions as far as establishing an incomplete plan is concerned. The virtual action serves as a bridge to link other real actions together in an incomplete plan and identify information that is potentially missing. In the travel reservation example, a traditional planner can be used to generate a plan for attending a conference: book air ticket => book hotel => rent car. However, it will fail if, for instance, the booking hotel service is not available. In contrast, in this situation our planner can generate an incomplete plan consisting of real and virtual actions: book air ticket => **virtual action** => rent car. Therefore, our approach has broader impacts: it stands to benefit both domain experts and regular users. Specifically, it makes it easier for domain experts to tune the planning domain, and it suggests to a regular user not only what is available (i.e., the real actions in the plan) but also what could be missing (i.e., the virtual action).

In this feasibility study, we focus exclusively on deterministic planning, in which each action produces a single (i.e., deterministic) outcome, owing to the significant role they play in AI planning, i.e., not only is deterministic planning still an active research area in AI planning, it is still widely used in state-of-the-art software engineering fields, e.g., web service composition research [6, 7]. Moreover, the proposed approach can be extended to nondeterministic planning via techniques that enable the application of deterministic planners to nondeterministic planning problems [8]. To our knowledge, we are the first to provide a workable approach that can propose virtual actions in the event of planning failure, as well as metrics for evaluating the proposed approach. We believe that this lays the groundwork for researchers interested in exploring this new direction.

In the rest of the paper, we introduce the relevant background concepts in Section II, describe our method for returning incomplete plans in Section III, present experimental results in Section IV, and conclude the paper by identifying future research directions in Section V.

II. BACKGROUND

We first introduce the notations and definitions about deterministic planning that we will rely on in the rest of the paper. Then, we present two important AI concepts: planning graphs [9] and genetic algorithms [10], based on which our approach is built.

A. Deterministic Planning

Definition 1. A deterministic planning domain is a 4-tuple $\Sigma = \langle P, S, A, \gamma \rangle$, where:

- P is a finite set of propositions;
- $S \subseteq 2^P$ is a finite set of states in the system;
- A is a finite set of actions; and
- $\gamma: S \times A \rightarrow S$ is the state-transition function.

An action a in Σ consists of a precondition, $pre(a)$, and an effect, $eff(a)$. $eff(a)$ is composed of two parts: the add effect and the delete effect. The add effect will be added to the state to which a is applied and the delete effect will be removed from the state. For example, the action "move" can move a robot from one place to another. Suppose that in the current state the robot is at location A. Then, the action "move(A, B)" will generate the add effect of the robot being at B and the delete effect is the robot being at A.

Definition 2. A planning problem is a triple $\langle s_0, g, \Sigma \rangle$, where s_0 is the initial state, g is the goal condition, and Σ is the planning domain.

B. Planning Graph

A planning graph is a data structure that provides a search space employed by many deterministic planners to generate a plan. More specifically, a planning graph is a directed and layered graph *interleaved* with *proposition levels* and *action levels*. The first proposition level contains the initial state and the first action level consists of actions that are applicable to the first proposition level. The effects of the actions in the action level *together with the propositions* in the previous proposition level form the next proposition level. This definition enables the planning graph to be extended in the forward direction until it has achieved a necessary (but perhaps insufficient) condition for plan existence. This corresponds to the *graph expansion* phase [9].

Due to the way of graph expansion, if a proposition appears in the current proposition level, it will appear in the rest of the proposition levels. In other words, every proposition level contains all the propositions that appear in the previous proposition levels. Thanks to this attribute, a planning graph has a special feature, level-off, which we will exploit in our approach. Level-off occurs when two adjacent proposition levels are identical but the goal condition has not been reached. Hence, level-off implies planning failure, since a fixed point is reached before any path to the goal is established. When level-off happens, the last proposition level contains *all* the propositions reachable from the initial proposition level (i.e., the initial state), thus representing the farthest level from the initial state.

C. Genetic Algorithms

A genetic algorithm (GA) is a population-based stochastic optimization search approach that has been widely applied in various research areas to solve problems that may not have polynomial time solutions [11]. Each state in its search space is known as a chromosome. The algorithm starts with an initial population of chromosomes. The aim is to iteratively create better and better populations by applying operators to "evolve" the chromosomes in a population so that eventually it finds one that is good enough to be used as a solution to the search problem.

Figure 1 illustrates the major steps of a GA. As mentioned, a population consists of a certain number of chromosomes, each of which is represented as a string that encodes a candidate solution to the given search problem. GA begins with a randomly generated population (line 1). To generate a new population, chromosomes are first selected from the current population based on their fitness values computed by a fitness function (line 2). Pairs of these

selected chromosomes are then randomly combined via the crossover operation to create new "child" chromosomes. Finally, each "child" chromosome is randomly "mutated" via the mutation operation (line 3). The GA algorithm terminates either when the maximum number of iterations has been reached (line 4) or a satisfactory solution (chromosome) has been found (lines 5 and 6). Otherwise, the new population is used in the next iteration (line 7).

1. Create an initial population of randomly generated chromosomes
2. Perform selection on the population based on the fitness values evaluated by a fitness function
3. Perform crossover and mutation on the selected chromosomes to produce the child population
4. If the max number of generations is exceeded, return the fittest chromosome
5. If any chromosome has a fitness value greater than or equal to the fitness threshold
6. return the chromosome
7. Otherwise, return to step 2

Figure 1. Outline of a Genetic Algorithm

III. METHOD

To help convey the proposed approach, we use a largely simplified travel reservation example as our running example. The three actions (services) in this domain are listed in TABLE I. Assuming that the user knows the travel dates and has the flight number provided by the travel agent, he wants to book a flight, a hotel, and a shuttle bus to the hotel, i.e., s_0 = {has_flt_num, has_dates} and goal g ={flt_booked, ht_booked, st_booked}.

TABLE I ACTIONS

Action	Precondition	Effect
Book_Flight	has_flt_num, has_dates	flt_booked, has_flt_info
Book_Hotel	has_flt_info, has_dates	ht_booked, has_ht_info
Book_Shuttle	has_flt_info, has_ht_info, has_dates	st_booked

It should not be difficult to see that a traditional planner can generate a plan "Book_Flight, Book_Hotel, Book_Shuttle". Assuming that the action "Book_Hotel" is missing from the domain, the traditional planning process will fail. We illustrate the planning failure in Figure 2. The missing action divides the correct plan into two halves. The reason why existing planners cannot provide useful information in case of planning failure is that they start from either s_0 or g, and plan toward the other end, but due to the missing action, the planning process will never reach the other end.

To overcome the above issue, in [12] we outlined a bi-directional search algorithm. However, no solutions were provided to implement the algorithm and no evaluations were conducted to investigate its feasibility. In this study, we intend to implement the algorithm and conduct experiments to evaluate the feasibility of the proposed approach. Specifically, the algorithm takes three major steps to return an incomplete plan when the traditional planning fails on a planning problem $\langle s_0, g, \Sigma \rangle$.

(1) A forward planning process starts from the initial state s_0 and proceeds as far as possible toward the goal condition g until it reaches the farthest place p_f.

(2) The second step starts a backward planning process from goal g to the initial state s_0 until it reaches the farthest place p_b.

(3) Finally, the third step suggests a virtual action a_v linking p_f and p_b together so that we can generate an incomplete plan composed of both a_v and real actions using a traditional planner.

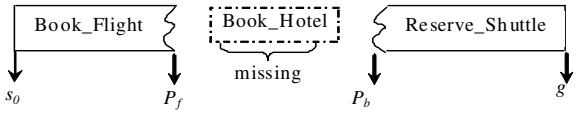

Figure 2: Illustration of Planning Failure

Two questions naturally arise. First, how can we determine the farthest places p_f and p_b? Second, how can we create a virtual action to link p_f and p_b together?

A. Determining p_f and p_b

To address the first question, we use the planning graph's intrinsic feature, level-off, to determine the farthest levels p_f and p_b. Recall that level-off occurs when all the possible actions have been applied to the planning graph but the goal condition still cannot be reached. Therefore, the proposition level at which level-off occurs represents the farthest level from the initial state. Given a planning problem $\langle s_0, g, \Sigma \rangle$, p_f is simply the last proposition level when level-off happens. To determine p_b, the farthest level in the backward planning process, we construct the planning graph based on the reversed planning problem $\langle g, s_0, \Sigma^{-1} \rangle$, in which g serves as the initial state and s_0 serves as the goal, and the preconditions and effects of actions in Σ^{-1} are the effects and preconditions of the corresponding actions in Σ. In the rest of the paper, we use p_f and p_b to denote the two proposition levels in which the forward and backward planning processes level off respectively.

In the travel reservation example, the forward planning graph levels off after applying the action "Book_Flight" to the initial state s_0, and p_f = {has_flt_num, has_dates, flt_booked, has_flt_info}. Note that "Book_Shuttle" cannot be applied because its precondition is not satisfied. Similarly, the backward planning graph levels off after applying the action Book_Shuttle^{-1}, and p_b = {flt_booked, ht_booked, st_booked, has_ht_info, has_dates}. Here, the original goal g becomes the initial state in the backward planning and the only applicable action is Book_Shuttle^{-1}, which is obtained by switching the original precondition and effect of the action Book_Shuttle.

B. Proposing a Virtual Action

We propose a virtual action a_v to link the two farthest levels p_f and p_b. Proposing a_v amounts to specifying its precondition, $pre(a_v)$, and its effect, $eff(a_v)$. We determine $pre(a_v)$ from p_f, and $eff(a_v)$ from p_b. At first glance, it seems that we can simply set $pre(a_v)$ to p_f, and $eff(a_v)$ to p_b. However, this will not work. p_f and p_b normally contain a large number of propositions. For example, it is not uncommon for p_f or p_b to contain more than 100 propositions, most of which are irrelevant to the missing part, and some of which are even mutually exclusive (e.g., a proposition and its negation). Including irrelevant propositions will make it difficult for a user to identify from $pre(a_v)$ and $eff(a_v)$ what the missing information in the domain is and understand why planning fails; and including mutually exclusive propositions does not even result in a valid action. Hence, we want to include only the relevant subset of propositions that are not mutually exclusive. However, for 100 propositions, there are 2^{100} ways to derive the possible precondition or effect for action a_v. It is therefore impractical to exhaustively enumerate them to find the best fit.

To deal with the huge number of candidate solutions, we first focus on propositions that are only available in the forward planning or the backward planning but not both. Specifically, let $P_{pre} = p_f - p_b$ and $P_{eff} = p_b - p_f$, where P_{pre} is the set of propositions that can only be obtained in the forward planning process and P_{eff} is the set of propositions that can only be obtained in the backward planning process. The goal is to find $pre(a_v)$ from P_{pre} and $eff(a_v)$ from P_{eff}. Hence, a_v will contain the essential information that is absolutely necessary to bridge the gap between p_f and p_b.

In our running example,

$P_{pre} = p_f - p_b$ = {has_flt_num, has_dates, flt_booked, fhas_flt_info} – {flt_booked, ht_booked, st_booked , has_ht_info, has_dates} = {has_flt_num}, and

$P_{eff} = p_b - p_f$ = {ht_booked, st_booked, has_ht_info}.

Let $pre(a_v) = P_{pre}$ and $eff(a_v) = P_{eff}$, then the virtual action has recovered the missing information.

Since the running example is largely simplified, we also used challenging and complex benchmark problems from International Planning Competitions (IPCs) [1] to evaluate the above approach. Our results show that the size of P_{pre} is about 10, which is reasonable for humans to comprehend. However, the size of P_{eff} may be still large (usually > 70), so it is still impractical to exhaustively enumerate all subsets of P_{eff} to compute $eff(a_v)$. Consequently, we propose to compute $eff(a_v)$ by using GA.

C. Using GA to Determine the Effect of the Virtual Action

As discussed in Figure 1, to use GA, we need to encode the chromosomes, design the fitness function, and define operators for selection, crossover, and mutation.

[1] All the domains can be found from http://www.icaps-conference.org/index.php/Main/Competitions.

1) Encoding a chromosome. We encode a chromosome as a binary string. Each bit in the chromosome corresponds to a proposition in P_{eff}. Specifically, the propositions in P_{eff} are indexed such that the first proposition corresponds to the first bit of the binary string, the second proposition corresponds to the second bit, etc. If a proposition is selected to be included in the effect of the virtual action a_v, the corresponding bit of the binary string is set to 1; otherwise, the bit is 0. To ensure that a chromosome does not contain mutually exclusive propositions, we ensure that a proposition and its negation cannot be both 1 in the binary string. In our running example, assuming that P_{eff} is indexed as {ht_booked, has_ht_info, st_booked}, the chromosome "110" denotes the subset {ht_booked, has_ht_info} because the bit corresponding to "st_booked" is 0 and therefore is excluded from the subset.

Given this encoding scheme, we can create an initial population of chromosomes, where each chromosome corresponds to a random subset of propositions in P_{eff}. To generate the next population, we need to compute fitness value of each chromosome.

2) Designing a fitness function. We compute the fitness value of a chromosome as follows. For each chromosome c, we create a virtual action $a_v(c)$, where $pre(a_v(c))$ is simply P_{pre}, and $eff(a_v(c))$ is the subset of the propositions that c contains. Using $a_v(c)$ together with the real actions in the given domain, we attempt to generate an incomplete plan using a traditional planner. If the planner finds a plan, then we compute the fitness value of c as the size of this plan. Otherwise, we set its fitness value to -1. In essence, we consider a chromosome fitter if it results in a longer plan.

Why does it make sense to consider a chromosome fitter if it yields a longer plan? Note that a longer plan implies that more real actions are used, and that the role played by the virtual action is smaller. This is a greedy strategy, which hopes to make small the gap bridged by the virtual action and make easier for a human to pinpoint exactly what information is missing from the domain. In other words, chromosomes that yield longer plans correspond to virtual actions that are potentially more informative for a human. Note that we can conceive counter examples, in which virtual actions leading to shorter plans are more informative than those leading to longer plans. Nevertheless, our experimental results showed that this strategy worked well in most of the cases.

However, another issue arises: it may be time consuming or even impractical to run a traditional planner to obtain a plan for each chromosome, especially when the population size or the number of GA iterations is large. To address this issue, we use relaxed plans [13] to improve efficiency. Relaxed actions and relaxed plans are widely used in heuristic search. Relaxed actions ignore their delete effects. Therefore, no two actions are mutually exclusive with each other. As a result, a single forward planning graph expansion process is enough to efficiently obtain a relaxed plan, if any. Correspondingly, we redefine the size of a relaxed plan as

the number of relaxed actions minus the number of occurrences of the virtual action. In other words, we compute the fitness value of a chromosome as the number of real actions in the relaxed plan. Note that we may still find cases where a smaller sized relaxed plan contains more useful information than the larger ones. However, our experimental results show that this fitness function is very efficient and works well in practice.

3) Specifying the GA operators. The operators we employ for selection, crossover, and mutation are standard. As the selection operator, we use roulette-wheel, which probabilistically selects chromosomes according to their fitness values. Specifically, each chromosome c_i is selected with probability p_i, where p_i is computed by dividing the fitness value of c_i by the sum of the fitness values of all the chromosomes in the current population. As a result, the diversity of the population can be maintained since the weak chromosomes still have chances of being selected.

As the crossover operator, we employ one-point crossover (see Goldberg 1989 [10]), where the crossover point is created randomly. Finally, the mutation operator, which is applied to the newly generated population to maintain the genetic diversity, operates by flipping the values of arbitrary bits according to a predefined mutation probability.[2]

Below we describe two improvements to our approach, with the goal of enabling it to be effectively applied in practice.

D. Improvement 1: Relaxing Precondition Matching in the First Action Level of the Backward Planning

To motivate this improvement, let us make an observation. If an action a can contribute to the goal condition g (i.e., if $eff(a) \wedge g \neq \phi$) in the forward planning process $\langle s_0, g, \Sigma \rangle$, then it is natural for us to expect its reversed action a^{-1} to be included in the plan for the backward planning problem $\langle g, s_0, \Sigma^{-1} \rangle$. However, as discussed in the next paragraph, this is not guaranteed in reality. What this implies is that some useful information may be lost in the backward planning process.

The question, then, is: why is it not necessarily the case that a^{-1} is included in the backward plan if a appears in the forward plan? The reason is that the precondition matching scheme leads to the information loss. More specifically, to apply an action a to the initial state s_0, we require that $pre(a) \subseteq s_0$, where $pre(a)$ denotes the precondition of action a. However, such a precondition matching scheme is too strict for backward planning. The following example illustrates why this is the case. Assume that the goal condition is to "hold a container", i.e., (*holding hand container*), and that there is an action, grasp-container, whose effect is "(*holding hand container*) $\wedge$ (*not* (*handempty hand*))". Hence, action grasp-container can be used to contribute to the goal. However, in the backward planning problem, the original goal condition (*holding hand container*) becomes the initial state. The effect of action grasp-container becomes the precondition of its reversed action, i.e., grasp-container^{-1}. However, the state g cannot satisfy the precondition of grasp-container^{-1} because it has an additional proposition, i.e., (*not* (*handempty hand*)). Therefore, grasp-container^{-1} will not be picked in the backward planning.

To prevent such information loss, we relax the precondition matching condition in the first action level in the backward planning process. Formally, given a backward planning problem $\langle g, s_0, \Sigma^{-1} \rangle$, an action a^{-1} can be selected in the first action level if $pre(a^{-1}) \wedge g \neq \phi$. Then, from the second level, we resume the regular precondition matching scheme.

E. Improvement 2: Recursive Use of GA

There are two issues involved in the proposed approach that we have eluded so far. First, P_{eff} is obtained by $p_b - p_f$. We find that it is not unusual that some useful information is removed by this deduction operation. Since we run GA based on P_{eff}, all the removed useful information will be permanently lost. Second, GA can return a subset of propositions, whose size is usually in the range of one third to one half of the original proposition set. However, the resulting subset can be still large. For example, if the size of the proposition set is 120, then the size of the resulting subset is still about 40 to 60.

To address the first issue, we employ two steps. First, we run GA on P_{eff} and obtain a subset P_{eff}' of propositions, whose size is about half of the size of P_{eff}. Second, we put the removed propositions back in P_{eff}', i.e., $P_{eff}' = P_{eff}' \cup (p_b \cap p_f)$. Here, the first step makes room for the second step because the planner we used (i.e., FF [13]) has a size limitation on the number of propositions that an action can have. Nevertheless, the problem with these two steps is that the size of P_{eff}' can be large, which is the second issue listed above.

To address the second issue, we recursively apply GA to the resulting subset of propositions to return a sub-subset of propositions. Specifically, the propositions in the resulting subset are ordered. The chromosome encoding scheme remains the same, i.e., if a bit is "1", the corresponding proposition is included in the sub subset. Otherwise, it is excluded. Meanwhile, the selection, crossover, and mutation operators remain the same as before.

The question, then, is: how many levels of recursive calls of GA should we make? In our implementation, we limit the depth of the recursive calls of GA to 3. The reasons are two-fold. First, an excessive number of calls of GA may result in loss of useful information because the probability of selecting the right subset of propositions from a large set is small. Each call of GA will result in the loss of useful information. Second, the execution of GA is time consuming. Hence, the excessive calls of GA may

[2] Since we focus on demonstrating the feasibility of our approach, we do not fine-tune the parameters to achieve optimal results. This means that our approach could be improved with more sophisticated operators.

significantly prolong running time, which may reduce the practicality of our approach.

IV. EVALUATION

Next, we evaluate our approach. All the experiments were conducted on a desktop computer with an Intel Pentium-4 3GHz processor and 1 GB of memory. The operating system is Linux.

A. Problem Domains

The problem domains, Barman [BM], PSR, ebookstore [EB], and Openstacks [OS], are obtained from the deterministic tracks of the international planning competitions (IPCs). These domains were chosen because (1) they are meaningful in the real-world; and (2) they do not contain quantifiers, e.g., forall, exists, etc., since our system does not support quantifiers at the moment. Specifically, in Barman, a robot barman is responsible for manipulating drink dispensers, glasses, and a shaker. The goal is to find a plan of the robot's actions that serves a desired set of drinks. PSR specifies an electricity network that may be faulty. Depending on the states of the switches and electricity supply devices, the flow of electricity through the network is given by a transitive closure over the network connections at any point in time. Openstacks describes an NP-hard problem. A manufacturer may have many orders. Each order consists of different products, which can only be made one at a time. The goal is to have all the orders shipped with a minimum number of stacks, which are the temporary space to hold products. Ebookstore is even closer to reality. It depicts the OWL [14] Web services for an ebookstore. The ebookstore scenario includes the electronic purchase of a book. The user provides a book title and author, credit card information and the address that the book will be shipped to, and requires a charge to credit card for the purchase, as well as information about the shipping dates and the customs cost for the specific item.

B. Evaluation of the Proposed Virtual Actions

To evaluate our approach, we removed one or two action(s) at a time from the benchmark domains. The removed actions were involved in the plans to the planning problems. In other words, the removal of these actions will result in planning failures. Although we can experiment with removing even more actions, we do not know exactly how many actions are missing in reality when a planning process fails. In addition, assume that the solution to a planning problem is a_1, a_2, …, a_m, a_{m+1}, …, a_{n-1}, a_n, a_{n+1}, …. If we remove the actions a_m and a_n, then the actions in between a_m and a_n are also indirectly removed. Hence, removing two actions should be sufficient for us to mimic a realistic planning failure situation.

TABLE II shows the number of actions we tested for each domain. For example, for BM, 11 of the experiments involve removing a single action, and 7 experiments involve removing a randomly selected pair of actions. For PSR, there are 50 domain-problem sets, each of which consists of a domain and a problem. We used the first set, in which five actions were used to solve the problem. Each of the 5 actions was tested in the experiments for removing a single action and 6 experiments involved removing randomly selected action pairs. For openstacks, there are more than 100 actions, which can be classified into 5 categories, namely, make product, open new stacks, start order, set up machine to make product, and ship the order. If we remove one action, the planning will not fail because the planner can find an alternative plan by using other actions in the same category. Therefore, we removed the whole category each time to perform our study and treated the whole category as a single composite action. For ebookstore, 6 experiments were for evaluating the removal of a single action and 4 experiments were for removing two actions.

TABLE II NUMBER OF ACTIONS TESTED FOR EACH DOMAIN

	BM	EB	OS	PSR
1 action	11	6	5	5
2 actions	7	4	3	6

As noted above, in each experiment we remove one or two actions, and then run our algorithm to propose a virtual action. Since our algorithm focused on determining the effect $eff(a_v)$ of a virtual action a_v, one way to evaluate a_v is to determine how many propositions in the effects of the action(s) being removed appear in $eff(a_v)$. Note, however, that some propositions in the effect of an action are more important than the others. For example, the grasp container action in barman aims to pick up the container from the table. Hence, the proposition (*holding hand container*) is important, whereas other propositions, such as (*not* (*handempty hand*)), are less important because without "holding the container" the rest of the propositions are meaningless. We identify for each action the important propositions and refer to them as key propositions. Since key propositions are important, we first focus on evaluating for each removed action (or action pair) whether $eff(a_v)$ contains all, some, or none of the key propositions appearing in its effect.

The results are shown in TABLE III. The column "Completely" shows the number of actions for which the key propositions are completely recovered by the virtual actions; "Partially" shows the number of actions where only some of the key propositions are recovered; "missed" shows the number of actions for which none of the key propositions are found. For 57% of the cases, the virtual actions recovered all of the key propositions; and for 21% of the actions, the virtual actions recovered some of the key propositions.

TABLE III EXPERIMENTS WITH RECOVERING KEY PROPOSITIONS

Domain	**Completely**		**Partially**		**Missed**		**Total**	
	1 act	2act	1act	2act	1act	2act	1act	2act
BM	7	2	2	3	2	2	11	7
EB	6	4	0	0	0	0	6	4
OS	3	1	0	2	2	0	5	3
PSR	3	1	0	3	2	2	5	6

While the non-key propositions are comparatively less important, they may still help domain experts gain a better

understanding of the domain and should ideally be recovered as well. As a result, we evaluate how well $pre(a_v)$ and $eff(a_v)$ match the precondition and effect of the removed action (or action pair), $pre(a_r)$ and $eff(a_r)$. We employ two evaluation metrics, *precision* and *recall*. Taking $eff(a_v)$ as an example, the precision of $eff(a_v)$ is the percentage of the propositions in $eff(a_v)$ that appears in $eff(a_r)$; and its recall is the percentage of propositions in $eff(a_r)$ that appears in $eff(a_v)$. In other words, a low precision implies that a_v contains many irrelevant propositions, and a low recall implies that a_v misses many correct propositions. Hence, it is desirable that both precision and recall are high. The precision/recall of $pre(a_v)$ can be computed similarly. For example, assume that the action of picking up container is missing in the BM domain. The effect of the corresponding virtual action includes (*holding hand container*) and (*clean shot*). Therefore, the virtual action recovers the key proposition (*holding hand container*), but misses the other two propositions, namely, (*not* (*ontable container*)) and (*not* (*handempty hand*)). In addition, the virtual action contains an irrelevant proposition, namely, (*clean shot*). Hence, the precision is 1/2 = 0.5 and the recall is 1/3 = 0.3.

TABLE IV PRECISION AND RECALL

	Precision				**Recall**			
	Precond		**effect**		**precond**		**effect**	
	1act	2act	1act	2act	1act	2act	1act	2act
BM	0.28	0.50	0.12	0.22	0.14	0.29	0.45	0.35
EB	0.67	0.50	0.58	0.67	0.67	0.67	1.00	1.00
OS	0.31	0.42	0.23	0.15	0.46	0.41	0.45	0.21
PSR	0	0.25	0.22	0.40	0	0.25	0.35	0.31
Avg	0.32	0.41	0.27	0.34	0.30	0.37	0.56	0.45

TABLE IV shows the precision and recall averaged over the virtual actions. On average, the actions recalled ≥ 30% of the preconditions and ≥ 45% of the effects, and have a precision of ≥ 32% (preconditions) and ≥ 27% (effects).

C. Human Evaluations

While the values shown in TABLE IV are useful for other researchers to compare against our results, it may not be easy to tell whether these values can be considered good or not. A more direct way to evaluate the usefulness of virtual actions is to employ humans, since ultimately virtual actions are meant to provide useful information for humans. Here is the setup of our human experiment. First, we educated 17 human participants consisting of 12 undergraduate students and 5 graduate students. All the participants had few or no knowledge about AI planning prior to enrolling in this study. For each domain, we prepared an instruction document for them to read, including the domain description, how to read the actions on the domain, and the list of actions defined on the domain. Second, we handed out 52 virtual actions to different human participants, and for each virtual action a_v, we asked them to identify all the real actions in the domain that they thought similar to a_v. We did not tell the participants whether the virtual actions were related to the removal of a single real action or two real actions. If they considered more than one real action as relevant to a virtual action, they were allowed to choose all of them, but they needed to rank the real actions in terms of relevance.

We score the human output as follows. If the evaluation is for one action, the response has a score of $1/n$ with n being the rank of the correct action. For example, the participant identifies two actions, a_{r1} and a_{r2}, that they think relevant to the virtual action a_v. He or she ranks a_{r1} to be more relevant than a_{r2}. Assume that the true action being removed is a_{r2}. Then, the score of the response is 1/2 since the rank of the correct action is 2 in the response. If the evaluation is for two actions, the response has a score of 1 (i.e., completely correct) if the first two actions are the right actions. Otherwise, the score is $1/m + 1/n$, where m and n are ranks of the right actions. If the response only includes 1 right action, the score is $1/2n$, where n is the rank of the right action.

TABLE V shows the human evaluation results. As we can see, the participants could easily locate the right actions on ebookstore because the precision and recall on this domain is the highest among the four domains (see TABLE IV). In contrast, they did a poor job on PSR because they lacked the knowledge of electricity network despite our instruction document. As a result, they could only guess the results. For BM and OS, they demonstrated good understandings of the domains, but some of them were overwhelmed by the large amount of information provided by the virtual actions on BM (note from TABLE IV that BM has a low precision). OS seemed to be a domain manageable by humans given its reasonable precision and recall.

TABLE V HUMAN EVALUATION RESULTS

Domain	**One Action**		**Two Actions**	
	# of judgments	avg. score	# of judgments	avg. score
BM	6	0.25	6	0.23
EB	8	1	7	0.51
OS	8	0.73	7	0.43
PSR	6	0.31	4	0.19
Avg	---	0.61	---	0.36

Although the average score for evaluations of two actions seem low, it is partly due to our rigorous scoring scheme. For example, some participants simply provided a single action as the answer. Even if the action is one of the right actions (i.e., precision is 1), the score will only be 0.5 according to our scoring scheme. We made a statistics on responses for two actions as shown in TABLE VI. The precision of the answers was 0.53 and 17 out of 24 answers (i.e., 71%) included at least one of the right actions. In summary, these encouraging results show the promise of our approach.

TABLE VI FURTHER ANALYSIS FOR EVALUATIONS OF TWO ACTIONS

Precision	Percentage of identifying at least one correct action
0.53	71%

D. Discussion

To evaluate the feasibility of our work, we proposed three different evaluation methods, namely, key propositions recovery, metrics of precision and recall, and human evaluation.

The use of key proposition recovery is easy to implement, but can be subjective since different people may identify different key propositions for the same action. In contrast, the metrics of precision and recall are objective. These metrics evaluate precisely how much useful information is recovered (i.e., recall) and the percentage of the useful information (i.e., precision) against the entire information including both relevant and irrelevant information. The issue with precision and recall metrics is that they did not tell us how well our approach did on the benchmark domains (see TABLE IV) since there is no related work to compare with. Our third evaluation method, human evaluations, compensates for this issue. The evaluation results directly showed how well humans could comprehend the virtual actions.

If we compare the results of the three evaluation methods (see TABLEs III, IV, and V), we can see that these results are consistent. For example, human participants performed well on the ebookstore (EB) domain, but did poorly on PSR. The virtual actions on EB completely (i.e., 100%) recovered the key propositions as shown in TABLE III, had the highest precision and recall as shown in TABLE IV, and received the highest evaluation scores as shown in TABLE V. In comparison, virtual actions on PSR constantly missed key propositions, suffered from low precision and recall, and received low human evaluation scores. Interestingly, as shown in TABLE IV, even though the precision and recall are 0 for the preconditions of virtual actions (i.e., 1 act) on PSR, human participants could still correctly figure out some of the virtual actions through their effects (as shown in TABLE V). This result can be explained by the fact that humans tend to put their primary focus on what the action can do rather than when the action can be applied.

V. Conclusion and Future Direction

We have investigated a new task that can significantly broaden the applicability of AI planning: generating an incomplete plan when missing information is present in a domain. We proposed the concept of virtual action, which serves as the bridge to link real actions. Preliminary results based on the IPC benchmark problems show that our approach holds promise. The generated virtual actions are human comprehensible. On average, 73% of the answers from the 17 research participants are partially or completely correct. In the next step, we intend to improve the fitness function of GA so that the fitness value can better reflect the useful information possessed by the virtual actions. We will continue to improve the precision and recall of the virtual actions to make our approach more practical to use.

Acknowledgment

This work was supported in part by Oklahoma Center for the Advancement of Science & Technology (OCAST) under grant HR12-036.

References

[1] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Hierarchical GUI Test Case Generation Using Automated Planning," *IEEE Trans. Softw. Eng.,* vol. 27, pp. 144-155, 2001.

[2] J. Fu, F. Bastani, and I. Yen, *Semantic-Driven Component-Based Automated Code Synthesis, Semantic Computing*: IEEE, Press/Wiley, 2010.

[3] L. A. Digiampietri, J. J. Pérez-Alcázar, and C. B. Medeiros, "AI Planning in Web Services Composition: a review of current approaches and a new solution," in *SBC 2007*, Rio de Janeiro, 2007, pp. 983-992.

[4] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," in *SWSWPC*, ed, 2004, pp. 43-54.

[5] W3C. (2002). *Web service use case: Travel reservation*. Available: http://www.w3.org/2002/06/ws-example

[6] A. Sirbu, A. Marconi, M. Pistore, H. Eberle, F. Leymann, and T. Unger, "Dynamic Composition of Pervasive Process Fragments," in *Web Services (ICWS), 2011 IEEE International Conference on*, 2011, pp. 73-80.

[7] X. Song, W. Dou, and J. Chen, "A workflow framework for intelligent service composition," *Future Generation Computer Systems,* vol. 27, pp. 627-636, 2011.

[8] U. Kuter, D. Nau, E. Reisner, and R. P. Goldman, "Using classical planners to solve nondeterministic planning problems," in *18th International Conference on Automated Planning and Scheduling (ICAPS)*, 2008.

[9] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artif. Intell.,* vol. 90, pp. 281-300, 1997.

[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Mass.: Addison-Wesley, 1989.

[11] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the Flexible Job-shop Scheduling Problem," *Computers & Operations Research,* vol. 35, pp. 3202-3212, 2008.

[12] J. Fu, W. Hao, M. Tu, B. Ma, J. Baldwin, and F. B. Bastani, "Virtual Services in Cloud Computing," in *IEEE 6th World Congress on Services (SERVICES 2010)*, Mimai, FL, 2010, pp. 467- 472.

[13] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," vol. 14, pp. 253-302, 2001.

[14] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, and T. Payne, "OWL-S: Semantic markup for web services," 2004.