Finding Inter-Related Plans

Biplav Srivastava
IBM India Research Laboratory
IIT, New Delhi 110016, India
sbiplav@in.ibm.com

Subbarao Kambhampati*
Arizona State University
Tempe, AZ, USA 85287
rao@asu.edu

Minh Binh Do
Palo Alto Research Center
Palo Alto, CA 94304
minhdo@parc.com

Tuan A. Nguyen
University of Natural Sciences
Ho Chi Minh, Vietnam
natuan@fit.hcmuns.edu.vn

Abstract

In many planning situations, a planner is required to return a set of plans satisfying the same goals which will be used by the external systems collectively. The external systems can specify the desired inter-relationships among the returned plans (e.g., diverse plans, similar plans, non-dominated plans) and the task of the planner is to return a set of plans which will meet these requirements. As an example, in adaptive web services composition, the web service engine wants to have a set of diverse plans/ compositions such that if there is a failure while executing one composition, an alternative may be used which is less likely to be failing simultaneously. In this paper, we investigate the problem, propose functions for defining similarity among plans and propose methods to find sets of inter-related plans.

Introduction

A typical AI planner takes as input the specifications of the initial and goal states and the set of available actions, and finds a plan that will satisfy the goals by efficiently searching in the space of possible states configurations or action orderings (plans). In many planning situations, a planner is required to return not one but a set of plans satisfying the same goals which will be used by the external systems collectively. The external systems can specify the desired inter-relationship among returned plans (e.g., diverse plans, similar plans, dominated plans) and the task of the planner is to return a set of plans which meet these requirements.

As an example, in adaptive web services composition, the web service engine wants to have a set of diverse plans/ compositions such that if there is a failure while executing one composition, an alternative may be used which is less likely to be failing simultaneously. However, if a user is helping in selecting the compositions, the planner could be first asked for a set of diverse plans and when she selects one of them, the planner is next asked to find plans that are similar to the selected one. Another example is using planning for intrusion detection (Boddy *et al.* 2005), where the aim is to detect as many ways of possible intrusion as possible where an intrusion attack is represented as a plan. A third, more general example involves any complex planning situation where the user is interested in optimizing multiple and possibly conflicting objectives, to generate a set of desired plans.

Existing planners, that are designed to find single solution plans, are not well suited for this problem. Even though many of the planners are capable of outputting multiple solutions by continuing their search beyond the first solution, they cannot guarantee any desired relations between the plans that are output.

To find inter-related plans, we need to be able to (1) define distance measures between plans and (2) modify existing planners so that they can use this distance measure to generate sets of inter-related plans. Similarity and diversity are examples of such inter-relationships for plans.

There has been very little work on this problem in planning. Hebrard et al 2005 solve the problem of similar/ dissimilar solutions for CSPs. If we consider their work for planning, since a planning problem of finite length can be compiled as a CSP problem, their results are the lower bounds for finding similar or diverse plans.

Our major contributions in the paper are:

- We formalize the problem of finding diverse/ similar plans by extending previous formulations for CSPs.
- We introduce useful bases and measures for plan distance. We show that different measures can give drastically different picture about inter-plan relationships.
- We discuss some preliminary work on effective solutions to the proposed problems.

We start by formalizing the problem and then propose a series of plan similarity function. Next, we propose methods to find inter-related plan and present initial results about their effectiveness. We then explore the problem with hierarchical plans. We end with discussion on related work and provide pointers for future work.

Problem Statement

At its simplest, a planning problem PP is a 4-tuple $\langle P, I, G, A\rangle$ where P is the set of predicates, I ($\subseteq P$) is the complete description of the initial state, G ($\subseteq P$) is the partial description of the goal state, and A is the set of executable (primitive) actions. A specification of an action consists of preconditions ($A_i^{pre} \subseteq P$) and postconditions ($A_i^{post} \subseteq P$).

*Kambhampati's research is supported in part by the NSF grant IIS-0308139 and the ONR Grant N000140610058.

A plan for PP is an action sequence S_j, such that if S_j is executed in I, the resulting state of the world would contain (entail) G. It is a 3-tuple $\langle T, O, \mathcal{ST} \rangle$ where: T is the set of steps in the plan; T contains two distinguished step names t_0 and t_∞. $\mathcal{ST}$ is a symbol table, which maps step names to actions. (Note that multiple steps can be mapped to the same action.) The special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ is always mapped to `finish`. The effects of `start` and the preconditions of `finish` correspond, respectively, to the initial state and the desired goals of the planning problem. O is a partial ordering relation over T.

As an example, suppose a person in Las Vegas (LV) wants to plan a weekend and is considering to visit one or more of the Disneyland (DL) in Los Angeles, his friend in San Francisco (SF) or an event at San Jose (SJ), based on the cost of each choice and its relative utility. In Figure 1, two possible plans S_1 and S_2 are given.

We also see in the Figure that there can be different representations for a plan. For example, plans can be hierarchical consisting of non-primitive actions (tasks) which can be decomposed further into primitive/ executable actions or other non-primitive actions by one or more reductions. In the example, we had considered *Travel* actions as primitive. If we consider them as non-primitive, plans are trees of AND-OR nodes. In Figure 1, S_3^1 and S_3^2 are two possible reductions of the non-primitive action *Travel* between LV and DL. We will focus on plans with primitive actions but we note that plans in alternative representations could also be compared.

Figure 1: Examples of plans.

Let $\delta(S_i, S_j) \rightarrow [0, 1]$ denote a distance function between a pair of plans. A value of 0 represents complete similarity of plans while 1 represents complete diversity. Following the convention of (Hebrard *et al.* 2005), we define *max(*δ*,* **S***)* = $\max_{S_i,S_j \in \mathbf{S}} \delta(S_i, S_j)$ and *min(*δ*,* **S***)* = $\min_{S_i,S_j \in \mathbf{S}} \delta(S_i, S_j)$. Also, we define *max(*$\delta$*,* **S***,* S_j*)* = $\max_{S_k \in \mathbf{S}} \delta(S_j, S_k)$ and *min(*δ*,* **S***,* S_j*)* = $\min_{S_k \in \mathbf{S}} \delta(S_j, S_k)$. In Table 1, we summarize the notations that are followed and Table 2 lists the various problems to find inter-related plans. The first 3 problems, *d*DISTANT*k*SET, MAXDIVERSE*k*SET and MAX*d*DISTANTSET (and their respective *close* variants) are planning adaptations of offline CSP problems in (Hebrard *et al.* 2005) while MOSTDISTANT is adaptation of their online problem[1]. We also introduce the *nNEAR* variations of the offline problems which takes the planning problem and a reference (previous) plan as input and requires that all returned plans be close to the reference plan.

Notation	Description
PP	A planning problem
Plan(PP)	Set of all plans of PP
S_i, S_j	A plan for PP ($\subseteq$ *Plan*(PP))
S, S'	Sets of plans

Table 1: Notation used in the paper.

Problem	Description
*d*DISTANT*k*SET (resp. *d*CLOSE*k*SET) Input: PP; Output: S	Find S with $S \subseteq$ *Plan*(PP) $\mid S \mid = k$ and *min(*δ*,* **S***)* $\geq d$ (resp. *max(*δ*,* **S***)* $\leq d$)
MAXDIVERSE*k*SET (resp. MAXSIMILAR*k*SET) Input: PP; Output: S	Find S with $S \subseteq$ *Plan*(PP), $\mid S \mid = k$ and for all $S' \subseteq$ *Plan*(PP), $\mid S' \mid = k$, *min(*δ*,* **S***)* $\geq$ *min(*δ*,* S'*)* (resp. *max(*δ*,* **S***)* $\leq$ *max(*δ*,* S'*)*)
MAX*d*DISTANTSET (resp. MAX*d*CLOSESET) Input: PP; Output: S	Find S with $S \subseteq$ *Plan*(PP), *min(*δ*,* **S***)* $\geq d$ (resp. *max(*δ*,* **S***)* $\leq d$), and for all S' with *min(*δ*,* **S'***)* $\geq d$ (resp. *max(*δ*,* **S'***)* $\leq d$), $\mid S \mid \geq \mid S' \mid$
MOSTDISTANT (resp. MOSTCLOSE) Input: PP, S; Output: S_j	Find S_j with $S_j \in$ *Plan*(PP)-S, such that for S_k with $S_k \in$ *Plan*(PP)-S, *max(*δ*,* S*,* S_j*)* $\geq$ *max(*δ*,* S*,* S_k*)* (resp. *min(*δ*,* S*,* S_j*)* $\leq$ *min(*δ*,* S*,* S_k*)*)
*n*NEAR*d*DISTANT*k*SET (resp. *n*NEAR*d*CLOSE*k*SET) Input: PP, S_k; Output: S	Find S such that it is *d*DISTANT*k*SET (resp. *d*CLOSE*k*SET) and $\delta(S_i, S_k) \leq n$ for all $S_i \in S$
*n*NEARMAXDIVERSE*k*SET (resp. *n*NEARMAXSIMILAR*k*SET) Input: PP, S_k; Output: S	Find S such that it is MAXDIVERSE*k*SET (resp. MAXSIMILAR*k*SET) and $\delta(S_i, S_k) \leq n$ for all $S_i \in S$
*n*NEARMAX*d*DISTANTSET (resp. *n*NEARMAX*d*CLOSESET) Input: PP, S_k; Output: S	Find S such that it is MAX*d*DISTANTSET (resp. MAX*d*CLOSESET) and $\delta(S_i, S_k) \leq n$ for all $S_i \in S$

Table 2: Different instances of the inter-relationship aware planning problem.

Distance Measures

In this section, we motivate different bases for comparing plans, the different methods of comparing plans, and propose useful plan distance functions.

Different Bases for Plan Comparison

At the heart of the problem of finding inter-related plans is the issue of defining criteria by which two plans are compared. A plan can be characterized by:

1. Actions that are present in the plan
2. Its behavior where the behavior represents the set of states that an execution of the plan will take
3. Causal chains that support the different goals achieved by the plan. They represent a middle-ground between actions and states by encoding how actions contribute to the goal states being achieved.

[1]We have converted all the decision problems to seek their solutions.

Basis	Pros	Cons
Actions	Does not require problem information	No problem information is used
States	Not dependent on any specific plan representation	Needs an execution simulator to identify states
Causal chains	Considers causal proximity of state transitions (action) rather than positional (physical) proximity	Requires domain theory

Table 3: The pros and cons of different bases to characterize plans.

These different criteria for characterizing plans can also serve as the basis for different ways of comparing plans. Table 3 gives the pros and cons of using the different basis. We note that if actions in the plans are used as the basis for comparison, no problem or domain theory information is employed. If plan behaviors are used as the basis for comparison, the representation of the actions that bring about state transition becomes irrelevant since the actual states that an execution of the plan will take is considered. Hence, we can now compare plans of different representations, e.g., 4 plans where the first is a deterministic plan, the second is a contingent plan, the third is a hierarchical plan and the fourth is a policy encoding probabilistic behavior. If causal chains is used as the basis for comparison, the causal proximity among actions is now considered rather than just physical proximity in the plan. But it requires the domain theory to be available.

Different Ways for Computing Comparison

After a basis for plan comparison is chosen, there can be different ways of using its characterizer to derive distance functions. The analogy we use is to string comparison where inter-relationship among characters in the string are used to define distance functions. Without loss of generality, assume that we are interested in action based comparison of plans.

One way to measure distance between plans is to consider plans as sets of actions and string similarity functions that depend on characters sets. This view ignores the absolute position of an action in the plan string and cares only about the presence or absence of an action in the plan.

Set-difference based Distance Computation We can also use the set-difference measure between plans. Here, the distance between plans S_i and S_j is measured as the number of actions that occur in one plan but not the other. This measure is used in (Myers 2005; Fox *et al.* 2006).

$$\delta_1(S_i, S_j) = \frac{\mid S_i - S_j \mid + \mid S_j - S_i \mid}{\mid S_i \mid + \mid S_j \mid} \quad (1)$$

Neighbourhood-based Distance Computation We can also consider the ordering of the actions in the plan (characters in the string). Let S_i and S_j be broken into substrings **P** = $P_1, ..., P_K$ and **Q** = $Q_1, ..., Q_L$. Then neighbourhood similarity functions follow the general pattern as follows.

$$\delta_*(S_i, S_j) = \frac{1}{K} \sum_{i=1}^{K} min_{j=1}^{L} \delta^{'}(P_i, Q_j) \quad (2)$$

Name	Basis	Computation
δ_1	Actions	Set-difference
δ_2	Actions	Prefixes Neighbourhood
δ_3	States	Set-difference
δ_4	States	Prefixes Neighbourhood
δ_5	Causal Chains	Set-difference
δ_6	Causal Chains	Prefixes Neighbourhood

Table 4: A spectrum of distance functions based on different bases and way of computations.

where $\delta^{'}$ refers to some secondary similarity function. Let $\delta^{'}$ be δ_1 in the remainder. We give the distance function based on prefixes and more can be proposed based on how different substrings are created.

Prefixes-based Distance Computation **P** and **Q** contains prefixes of S_1 and S_2, respectively.

$$\delta_2(S_i, S_j) = \delta_*(S_i, S_j) \quad (3)$$

In Table 4, 6 distance functions are presented which use 3 different bases and 2 different ways of computation. More distance functions can be derived by extending any of the two dimensions.

Action	Preconditions	Effect
A1	p1	g1
A2	p2	g2
A2'	p2, g1	g2
A3	p3	g3
A3'	p3, g2	g3

Plan	Goal	Causal Chains
S6-1, S6-2	g1	I-p1-A1-g1
	g2	I-p2-A2-g2
	g3	I-p3-A3-g3
S6-3	g1	I-p1-A1-g1
	g2	I-p1-A1-g1,p2-A2'-g2
	g3	I-p1-A1-g1,p2-A2'-g2-p3,g2-A3'-g3

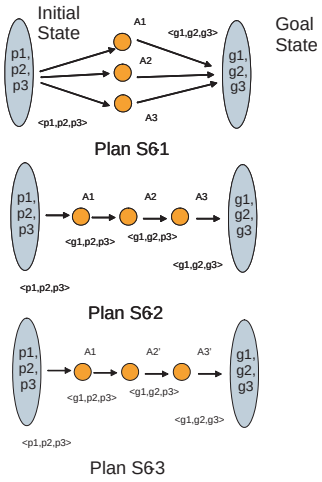

Figure 2: Three different plans for an example problem. The significant fluents of the states after every action is shown in $\prec$ ... $\succ$. The domain description and causal chains in the 3 plans are also given.

Example: Comparing Plans by Different Bases

In Figure 2, three plans are shown for a planning problem where the initial state is $\langle p_1, p_2, p_3 \rangle$ and the goal state is $\langle g_1, g_2, g_3 \rangle$. Plans S6-1 and S6-2 have the same actions but different structures. S6-1 has parallel actions while S6-2 has them in sequence. The plan S6-3 has A_1 like the other plans but all other actions are different ($A_2^{'}$ and $A_3^{'}$). However, it also achieves the same goals.

An action based plan comparison method which uses prefix-neighbourhood based distance computation would find S6-1, S6-2 and S6-3 to be all different. This is because

all the three plans have different sets of action prefixes. If instead, the action information is used with set differencing, S6-1 and S6-2 would be found identical.

A state based comparison method which uses any of the given computation choice would find S6-2 and S6-3 to be identical, and both of them to be different from S6-1. This is because the states after every transition in S6-2 and S6-3 are identical. S6-1, on the other hand, has (trivially) the same first and last states but no intermediate states.

A causal link based comparison method which uses set differencing would find S6-1 and S6-2 to be the same while S6-3 as different. The causal chains for all the goals are shown in the figure.

Solution Methods

We start by noting that one straightforward approach to generate k multiple inter-related plans is to make existing planners run further after finding the first solution. As new solutions are generated, their distance from the already selected plans can be assessed and used to decide whether they should be added to the selected set.

A more efficient alternative would be to bias the planners' search process so it progresses towards plans that are likely to meet both the solution quality and inter-releatedness constraints. There are broadly two ways of achieving this: One idea is to bias the search for the latter plans so as to satisfy the inter-relatedness constraints. In the case of heursitic search planners such as FF, this can be accomplished by modifying the heuristic to reflect the inter-relatedness constraints. For example, if we are interested in finding maximally dissimilar plans, we can penalize plans similar to the ones already selected. The technical challenge in implementing this approach would be making the heuristics sensitive to inter-relatedness constraints.

The heuristic search approach outlined above is "greedy" in the sense that the seed plans that we have already committed to could force us into a sub-optimal overall set of plans. The second idea is thus to search simultaneously for the k solutions. One avenue for doing this is to model planning as constraint satisfaction (c.f. (Do & Kambhampati 2001)), and adapt the technique proposed in (Hebrard *et al.* 2005) for simultaneously searching for k inter-related CSP solutions.

As of this writing, we have experimented with one specific implementation of each of the above ideas. We will describe the techniques implemented and provide preliminary results. We are currently in the process of completing a more careful investigation of the comparative advantages of these approaches.

Compiling Planning as CSP and Solving for Exact Diverse Plans

The GP-CSP planner(Do & Kambhampati 2001) is a Graphplan based planner that converts Graphplan's planning graph into a CSP encoding, and solves the CSP encoding using standard CSP solvers. Here, the variables correspond to the predicates that have to be achieved at a level and its possible values are the actions that can support the predicates. Constraints encode the relationship (e.g., mutual exclusions) among predicates and the relationship among the supporters of the predicates.

Similar to the way Hebrard et. al 2005 solved their *d*DISTANT*k*SET/*d*CLOSE*k*SET problem by reformulating it as a new CSP, we solve the same problem with different distance measures by making k copies of each planning encoding. Each encoding is created using GP-CSP planner and the k copies are connected to each other using global constraints. Due to the way the CSP library is used in conjunction with the planning graph structure to solve the planning encoding, there are some complications. The details of our approach are:

- As opposed to creating k(k-1)/2 special variables to represent the distances between each pair of copies, we create k(k-1)/2 global constraints connecting them. If each copy has n variable, then this constraint involves 2n variables from each of k(k-1)/2 possible pairs of k copies[2]. Each global constraint between the ith and jth copies ensures that two plans represented by the solutions of those two copies will be at least/most d diverse/similar to each other.
- Because the CSP library used in GP-CSP uses implicit constraint representation, we implement special constraint checking routine to check those k(k-1)/2 constraints. Those routines are called upon by the normal forward checking and arc-consistency checking procedure inside the default solver. In the future, we plan on implementing special consistency checking techniques to deal more efficiently with those global constraints.

Due to the special planning encoding in GP-CSP and the distance measure defined earlier in this paper, there are substantial differences between how each global constraint is satisfied between traditional CSP encoding as in (Hebrard *et al.* 2005) and our encoding. In our encoding, facts represent variables and actions represent values. A given action a can represent different values in domains of different variables. For example, if there are two variables x_1 and x_2 and their current assignments in the first copy are $\{x_1 \rightarrow v_1, x_2 \rightarrow v_2\}$ and in the second copy, they are: $\{x_1 \rightarrow v_2, x_2 \rightarrow v_1\}$, then in traditional CSP, the distance between two sets of assignments would be 2. However, the value v_1 of x_1 and v_1 of x_2 may represent the same action instance, also v_2 of x_1 and v_2 of x_2. Therefore, the distance between those two set of assignments in our planning encoding can be 2, 1 or even 0.

Thus, when each global constraint is called upon to check if the distance between two copies is within/over a predefined value d, we first have to map each set of assignments to an actual set of actions. Then, we compare the action sets (not the variable assignments) to decide if the two copies satisfy the global constraint defined by the distance measure. This process is done by mapping each variable $\rightarrow$ value into action using a call to the planning graph, which is outside but works closely with the general purpose CSP solver in GP-CSP.

[2] An alternative approach would be to create only one global constraint involving k*n variables from all k copies.

Problem	k	d	Time (in sec)	Dist.(Min, Max, Avg)
prob002-rocket-a	2	0.1	2.45	(0.154, 0.154, 0.154)
		0.2	6.72	(0.862, 0.862, 0.862)
		0.8	6.79	(0.862, 0.862, 0.862)
prob002-rocket-a	3	0.05	11.1683	(0.154, 0.862, 0.626)
		0.1	10.88	(0.154, 0.862, 0.626)
prob004-log-a	3	0.05	7.86	(0.054, 0.203, 0.151)
		0.1	24.11	(0.197, 0.698, 0.525)
		0.15	21.04	(0.197, 0.698, 0.525)
		0.2	19.50	(0.209, 0.701, 0.536)

Table 5: *Initial results of GP-CSP in* d*DISTANT*k*SET*

Table 5 presents the results of GP-CSP on some logistics problems as run on a Pentium-3 667Mhz with 256MB RAM. We see that this approach can give diverse plans effectively. The last column shows the diversity in the returned plans using δ_1 and that it is greater than minimum diversity needed for the problem (d). We also found that with higher k and d, the problems take longer to solve, as expected.

Heuristic Approach for Approximate Diverse Plans

In heuristic state space planning, a search framework like A* is used to find plans driven by heuristics that measure the progress to goals. Specifically, the cost of a search node is measured by:

$$f(s_i) = g(s_i) + w * h(s_i) \quad (4)$$

where g is the cost to achieve the current node starting from the initial search state, h is the heuristic estimate of the effort to achieve the goals and w is a weighing function. In measuring h, heuristics derived from the relaxed planning graph (RPG) have been found to be very effective(Nguyen, Kambhampati, & Nigenda 2002).

The RPG heuristic estimate can be biased towards plans that use as many of the actions already supporting other goals as possible. To do this, the planner will now take as input not just the goals to be supported, but also the set of actions already committed to in previous plans. RPG heuristic can also be biased to find plans that do not share many actions with another plan. Thus, the relaxed plan extraction process will be biased to avoid actions that are in the input plan. This approach works as long as we have similarity measures that are dependent on action presence and not on the relative position of the actions.

We implement this idea by using $h'(s_i)$ as defined below instead of $h(s_i)$.

$$h'(s_i) = h(s_i) + w_\delta * \delta_j(s_i, S_0) \quad (5)$$

Hence, we increase the heuristic values with a weighted factor accounting for the distance between the partial plan and the reference input plan. δ_j can be any distance measure including the ones defined earlier. If w_δ is positive, search nodes close to the reference plan (S_0) get priority over other nodes. If w_δ is negative, search nodes away from neighbourhood of the reference plan get priority. Given a distance function, one can start from a reference plan and control the relationship of the subsequent plan by using appropriate w_δ. The new plan can be added to the reference plans set and more plans generated appropriately related to it.

Problem	w_δ	w_k	Time (in sec)	Dist. (Min, Max, Avg)
bw-prob-4-0	-100	0	2.08	(0.334, 0.334, 0.334)
	10	100	2.04	(0.0, 0.0, 0.0)
	-10	-100	2.22	(0.156, 0.318, 0.249)
lilprob-4-0	-10	-10	1.77	(0.143, 0.334, 0.242)
	100	-100	1.68	(0.0, 0.0, 0.0)
	-100	-100	1.73	(0.0, 0.5, 0.334)

Table 6: *Initial results of Planner4J RPG for k=3*

The above approach would work regardless of whether the h was obtained from a relaxed plan of RPG (e.g., AdjSum2 heuristic) or not (e.g., max heuristic). We can also affect the relaxed plan extraction process for obtaining h. Consider the AdjSum2 heuristic where the value of a state is estimated by the length of the relaxed plan to reach the goal and an interaction factor derived from the maximum interaction among the predicates in the state.

$$h_{AdjSum2}(s_i) = len(RP(s_i)) + max_{p,q \in s_i} \Delta(p, q) \quad (6)$$

$$len(RP(s_i)) = len(RelPlan(s_i) + w_k * \delta_k(RelPlan(s_i), S_0)) \quad (7)$$

Specifically, at each step of the extraction of the relaxed plan, we prefer actions that are not present in other plans. We give a cost metric to each action in the PG and the cost is dependent on how many of the other plans these actions already support. Then we can use the costs to extract the relaxed plan.

We have implemented both the methods in the Planner4J family of Java planners and they together seem to give the best results. Table 6 presents the results on a sample of blocksworld and logistics problems as run on a Pentium-M 1.5GHz with 1.25GB RAM under Windows XP with w=5. We see in lilprob-4-0 that only w_k could not lead to diversity in the solution.

Related Work

Although the need for finding similar or different plans has been noticed in the past, there has been little concrete work on formalizing and solving the problem. Researchers including Tate (Tate, Dalton, & Levine 1998) and Myers (Myers 2005) have articulated the need for finding dissimilar plans. Myers, in particular, allows evaluating the plan similarity, but does not seem to provide a way of generating dissimilar plans efficiently. As we mentioned earlier, intrusion detection work by Boddy et. al. (Boddy *et al.* 2005) focuses on finding multiple qualitatively different plans for a problem. However, they coerce a traditional planner (Metric-FF) to generate multiple plans, and filter them out in a post-processing phase. Boddy et. al. acknowledge the need for a technique that takes inter-relatedness constraints into account during search more actively. The problem of finding similar plans has been considered in the context of replanning. A recent effort in this direction is (Fox *et al.* 2006), which shows how a local search planner called LPG can be modified to produce a plan that is similar to a reference plan.

Finally, Linden et. al. (Linden, Hanks, & Lesh 1997) motivate the need for finding related plans comprising a pareto set in the context of a travel planning scenario.

Outside of planning, we have already mentioned the connections to the work in CSP community in finding similar/dissimilar solutions. The challenges in finding interrelated plans also bears some tangential similarities to the work in information retrieval on finding similar or dissimilar documents (c.f. (Callan & Minka 2002)).

Conclusion and Future Work

In this paper, we investigated the problem of finding interrelated plans. We formalized the problem of finding diverse/similar plans by extending previous formulations for CSPs. We looked at the different bases for comparing plans, the different methods of computing comparison, and proposed useful plan distance functions. We conducted preliminary experiments with a CSP based exact approach and a heuristic based approximate approach to generate diverse plans. In future, we intend to implement more approaches and run extensive experiments.

References

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In *Proc. ICAPS*. AAAI.

Callan, J., and Minka, T. 2002. Novelty and redundancy detection in adaptive filtering. In *Proc. SIGIR*. ACM Press.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *AI* 132(2):151–182.

Fox, M.; Gerevini, A.; Serina, I.; and Long, D. 2006. Plan stability: Replanning versus plan repair. In *Proc. ICAPS*.

Hebrard, E.; Hnich, B.; O'Sullivan, B.; and Walsh, T. 2005. Finding diverse and similar solutions in constraint programming. In *Proc. AAAI*.

Linden, G.; Hanks, S.; and Lesh, N. 1997. Interactive assessment of user preference models: The automated travel assistant. In *Proc. UM*.

Myers, K. 2005. Metatheoretic plan summarization and comparison. In *Proc. ICAPS WK. Mixed-initiative Planning and Scheduling*.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search. In *AI*.

Tate, A.; Dalton, J.; and Levine, J. 1998. Generation of multiple qualitatively different plan options. In *Proc. AIPS-98, Pittsburgh*. AIAI.