Empirical Comparisons of Some Design Replay Algorithms

Brad Blumenthal †
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
brad@cs.utexas.edu

Abstract

Although most design replay techniques have been empirically tested against some performance program, there has been very little empirical evidence published that compares various approaches on the same problems to determine the source of power. Six different design replay algorithms based on approaches in the literature are implemented and tested on 20 different design replay problems. The resulting data indicate that there is a trade-off between efficiency and autonomy for certain types of adaptation strategies. Based on some of the lessons drawn from this data, a new algorithm, REMAID, has been developed. This algorithm recognizes two different types of mis-matches between previous experience and current problems: *detours* and *pretours*. The REMAID strategy takes advantage of its knowledge of mis-matches to improve replay autonomy without sacrificing efficiency. The success of the REMAID algorithm is empirically verified.

Introduction

Design replay has been proposed as a way of using previous design experience to improve the performance of automated design systems [Carbonell, 1986; Mostow, 1989]. Instead of attempting to reuse a previous *solution*, the design replay technique records the steps that went into producing a previous solution and replays the sequence of steps. This approach provides more flexibility in reusing experience by preserving intermediate steps of the problem solution and thus allowing partial reuse of the appropriate parts of the solution. However, this flexibility depends on a capability to adapt previous solution steps to fit a new problem.

Mostow enumerates the dimensions on which a design replay technique can be evaluated [Mostow, 1989]. This paper will concentrate on the dimensions of efficiency, or how much computation is required, and autonomy, or how much of the problem the replay algorithm addresses. An ablation study was performed using six algorithms that vary only in what strategies they use to adapt previous experience to fit a new problem. These strategies are based on replay strategies presented in the literature. The resulting data isolate and quantify the contributions of each of these adaptation strategies to replay efficiency and autonomy.

The data support several hypotheses. Some are unsurprising: design replay is generally an effective technique, and increased flexibility in adapting recorded experience to new problems increases both efficiency and autonomy. More informatively, the data also indicate that calling an automated design performance program to help adapt to replay failures can be a successful strategy, but that there may be a trade-off between efficiency and autonomy unless the execution of the replay program and the performance program are interleaved intelligently.

The lessons learned from this study have led to the development of a seventh algorithm: REMAID.[1] This algorithm increases both the efficiency and autonomy of the design replay process by appropriately interleaving execution of design replay and automated design. It does so by recognizing two types of replay failures that occur, *detours* and *pretours*, and by appropriately adapting design experience during replay. The REMAID algorithm was run on the design replay problems from the ablation study, and empirically, it performed with the best.

For clarity, the following terminology will be used in this paper: a design *episode* is a computation which takes a specification as input and produces a design as output. A design *experience* is a previously completed design episode. A design *goal* is a description of a design (sub-)problem and includes information about the entity being addressed and the desired outcome of addressing that goal. A *rule* is a heuristic used to select or

†Support for this research was provided by the National Science Foundation under grant IRI-8620052, Apple Computer Corp., and by the Army Research Office under grant ARO-DAAG29-84-K-0060.

[1] "MAID" is an acronym for "Metaphoric Application Interface Designer;" MAID is the performance component of this system which produces interface designs that resemble real world objects. "REMAID" is an acronym for "Replaying Episodes of MAID."

address a design goal. A design *history* is a recording of the goals and rules that make up a design experience. *"Step"* is short for a recorded goal and its associated rules and decisions. Thus, for the purposes of this paper, design replay is concerned with recording design experience and using the resulting design history to improve the efficiency and autonomy of a new design episode. Typically this is done by matching a goal in the history to a goal in the current episode, replaying the corresponding design rule, and continuing with the next step.

The Domain

To test the various replay algorithms, a performance program was written which automates the design of human interfaces for computer applications. The domain of automated human interface design is an especially appropriate one because interface design is expensive and time-consuming, yet it is often desirable to have multiple, alternate interface designs to a computer application. Such a collection of designs might be used to conduct user studies to determine what techniques are most effective or to tailor computer applications for specialized user communities.

The Performance Program

The performance program used for this study is the MAID interface designer. This program takes as input a frame-based description of the objects and operations that a computer program makes available to a user. It produces as output a design for a human interface to that application in terms of a set of interface techniques such as menus, graphics, text, *etc.* In addition, the MAID interface designer uses a knowledge base of real-world objects to produce metaphoric interfaces which mimic the appearance and behavior of such objects [Blumenthal, 1990].

One of the most expensive operations in the MAID program is choosing the next design goal to be addressed. A characteristic of the interface design domain is that there are default rules for addressing any design goal. As a consequence, MAID is not required to do any backtracking ; however, in order to achieve the best designs possible, MAID has to spend more effort ordering the design goals. Essentially, MAID trades backtracking for extra goal-ordering effort and the possibility of sub-optimal designs.

Therefore, at the heart of the MAID program is an *agenda* of pending design goals. MAID uses a set of ordering heuristics to select a goal from the agenda based on the features of each goal, what other goals are on the agenda, and the state of the design in progress. It then uses a separate set of design heuristics to address the goal.

Applying design replay techniques to this problem of selecting goals from the MAID agenda is promising for the following reasons: since there is no backtracking in MAID, the order in which design subgoals are addressed can significantly affect the resulting interface design; it is necessary to run the agenda ordering heuristics at every step, since the relevant factors can change in unpredictable ways during the course of a design; however, running all of the heuristics is expensive and only a small percentage of the heuristics tried at any step usefully reduce the size of the conflict set.

The Experiment

The MAID design program is currently capable of designing five interfaces to a single application.[2] Thus, there are five possible design histories, each of which can be used in the design of any of the five interfaces. This gives a total of 25 possible design replay episodes, including five trivial cases designing the same interface in both the recorded experience and the new episode.

The six algorithms summarized below were implemented and run on all 25 cases. Data were collected from the 20 non-trivial cases on the total number of goals addressed, the number of goals replayed from the recorded experience, the total number of ordering heuristics which were applied to the agenda, and the number of ordering heuristics that were useful in selecting the next goal to be addressed.[3]

Current Techniques

Mostow outlines the important issues in successfully applying the design replay technique, including acquiring design experience, determining the correspondence between previously and newly encountered design goals, and determining the appropriateness of a previously executed design step in a new situation [Mostow, 1989]. This paper will focus solely on the issue of adaptation: the problem of reusing design experience when parts of a new design episode do not match the previous experience.

There are two parts to this problem which Mostow refers to as the adaptation process. This paper departs from that terminology slightly. The first part of this problem, which is here referred to as "adaptation," is the question of what to do when a particular step in a recorded history fails to match or address a corresponding goal in the current design episode. Strategies for adaptation vary from simply ignoring the failing recorded goal to trying to modify the state of the current design episode so that the recorded step succeeds in finding or addressing a corresponding goal.

The second part is referred to as "recovery" and is the question of what to do when none of the goals in a recorded history can be successfully used to select a

[2] This is limited by the amount of knowledge entered in the knowledge base, not by any inherent limitations of the MAID program.

[3] Due to irrelevant technical details, accurate CPU times were unavailable. All of the ordering heuristics use similar amounts of CPU time, so for purposes of comparison, a count of heuristic applications is adequate.

corresponding goal from the current design state. Recovery strategies range from simply halting replay to invoking a performance program and then attempting to restart replay.

There have been a number of projects addressing the problems in design replay since Carbonell's derivational analogy proposal [Carbonell, 1986]. Three projects that the REMAID research builds on are the BOGART/VEXED project [Mostow, 1989], the work by Carbonell and Veloso on using derivational analogy in the PRODIGY system [Carbonell and Veloso, 1988], and the PRIAR project [Kambhampati, 1989a; Kambhampati, 1989b].

The following algorithms vary *only* in their approaches to adaptation. Since the domains used for research in these techniques have varied from circuit design to matrix manipulations to blocks world planning it has not been possible to reimplement exact versions of the various algorithms for use in the interface design domain. However, the approaches toward adaptation and recovery are domain-independent, and they have been preserved and implemented in order to determine how they affect the replay process.

Table 1 summarizes the six replay algorithms. There are two adaptation strategies: simply skip the recorded goal and try later or use alternate design rules. There are three recovery strategies: halt replay, use MAID for one goal then restart replay, or use MAID for all remaining goals. A short description of the research motivating each adaptation and recovery strategy follows.

BOGART/VEXED The replay program, BOGART [Mostow, 1989], records design goals and the design rules for achieving those goals as they are chosen by a designer who is using the VEXED circuit design tool. BOGART then applies the design rules in the order in which they were recorded. BOGART fails to reuse a design rule if the corresponding recorded goal fails to match a goal in the current episode or if the rule preconditions are not satisfied by the matching goal. The BOGART approach to adaptation is to simply skip any design step that fails to be replayed. It continues until it has tried every step remaining on the history without success. BOGART's approach to recovery is to halt replay and ask the human designer for a new history to replay.

HOBART[4] implements the BOGART approach to adaptation and recovery. HOBART attempts to match a goal from the history to a goal on the current agenda of pending goals. HOBART simply skips any step whose goal does not match a goal on the agenda or whose design rule does not address the matching goal. When no more goals on the history match any goals on the agenda, HOBART halts.

[4]The names of the reimplementations were chosen solely for their phonemic similarity.

PRODIGY Extensions Carbonell and Veloso's extensions to the PRODIGY system [Carbonell and Veloso, 1988] reuse a history that records the goal ordering decisions, variable bindings, and problem solving rule choices that produce a solution to a matrix manipulation problem, as well as the justifications for each of these decisions.

When PRODIGY fails to reuse a recorded step, then it follows what Carbonell and Veloso call the "satisficing approach" to adaptation and recovery. One strategy described for this approach is to address the matched goal in some other fashion and then continue replay at the next step. A second is to attempt to satisfy the violated justification(s) in some way and continue replay at the current step.

There are a number of ways these strategies can be implemented, so one simple adaptation strategy and one simple recovery strategy were developed and added to the basic HOBART algorithm, both separately and together. The adaptation strategy is based on the first satisficing strategy and is used when a recorded goal from the history matches a goal on the current agenda, but the recorded design rule does not apply. This strategy tries other design rules until one succeeds (recall from section that there is always a default rule which is guaranteed to succeed) and then continues replay at the next step.

The recovery strategy is invoked when all of the goals on the recorded history have been tried and none of them match a goal on the current agenda. In this case the MAID algorithm is invoked to select and address a goal on the agenda, and replay is started again where it was stopped. This is based on the second strategy; something is done to satisfy the justifications of some recorded goal and replay is started where it left off.

The implementation of HOBART with the PRODIGY-inspired adaptation strategy (alternate design rules) is called "PROBART." The implementation which has the recovery strategy (select and address one goal with a performance program) is called "POSSIBLY." The implementation with both the adaptation strategy and the recovery strategy is called "PROBABLY."

PRIAR Kambhampati's PRIAR system reuses plans produced by a non-linear planner to solve new planning problems [Kambhampati, 1989b]. PRIAR's adaptation strategy uses the pre- and post-conditions of the recorded plan operations to determine what operations in the recorded plan to reuse. This strategy is not feasible in the MAID domain.[5]

PRIAR's recovery strategy takes any goals that are not satisfied by the old plan and posts them as new goals that are then solved by the same non-linear plan-

[5]For one reason, the MAID domain has rules with conditional consequents. This prevents PRIAR from analyzing the recorded rules to determine whether they will have the desired effect.

Algortihm	*Adaptation Strategy*	*Recovery Strategy*
HOBART	Skip goal	Halt
PROBART	Alternate design rule	Halt
POSSIBLY	Skip goal	MAID for one goal; restart replay
PROBABLY	Alternate design rule	MAID for one goal; restart replay
BRIAR	Skip goal	MAID for all remaining goals
PYRE	Alternate design rule	MAID for all remaining goals

Table 1: Summary of algorithms tested: strategy for adapting to single recorded rule failures, and strategy for recovering when recorded history can suggest no more goals.

ner that produced the recorded plan. This strategy inspired a recovery strategy which use the MAID design program to address any goals left on the agenda without attempting to restart replay. The first, called "BRIAR," uses the HOBART algorithm until replay fails, and then invokes the MAID algorithm to address any left-over design goals. The second algorithm, called "PYRE," uses the PRODIGY-inspired adaptation strategy (using alternate design rules) and the PRIAR-inspired recovery strategy (turn control over to the performance program to finish the design).

Results

Table 2 presents the averages of the data collected from 20 different replay episodes of each algorithm. The data for MAID give the average number of goals addressed and the average number of heuristics tried and used by the MAID performance program over five designs and is presented for comparison.

The efficiency of the various algorithms is measured in terms of the number of applications of ordering heuristics that were needed to complete a design. All of the ordering and match heuristics measured use similar amounts of CPU time, so for purposes of comparison, searching the agenda for a goal matching a recorded goal on the history was counted as one heuristic application. Since all of the algorithms use ordering and match heuristics for adaptation and recovery, this metric adequately captures the overhead due to replay.

The number of heuristics tried represents the number of heuristics which were applied (successfully or not) in an attempt to select a goal from the agenda. The number of heuristics used represents the number of heuristics which successfully reduced the size of the conflict set (including those that actually chose a goal).

The autonomy of each algorithm was measured by the number of goals that it successfully replayed. All of the algorithms that used the MAID algorithm for recovery completed all of the designs. HOBART and PROBART halted when there were no more goals that could be replayed.

Both the PRODIGY-inspired adaptation plan (expand one goal and restart replay) and the PRODIGY-inspired recovery plan (try alternate design rules) added similar amounts of autonomy. This is evidenced by comparing the difference in goals replayed between HOBART and PROBART or POSSIBLY and PROBABLY (difference due to adaptation) with that between HOBART and POSSIBLY or PROBART and PROBABLY (difference due to recovery).

The PRODIGY-inspired adaptation strategy of using alternate design rules also contributed to efficiency. This is evidenced by the data for PROBART, PROBABLY, and PYRE. Since checking the preconditions of alternate design rules is very cheap compared to applying goal selection rules, the overhead for this approach is nominal.

In terms of efficiency, the PRODIGY-inspired recovery strategy was the most expensive, as seen by the results for POSSIBLY and PROBABLY. The most efficient performer is the PYRE algorithm which takes advantage of flexible adaptation with a less expensive recovery strategy. The least efficient is the POSSIBLY algorithm which used the more rigid adaptation strategy and the most expensive recovery strategy.

Discussion

The foremost lesson of the empirical data is that, in terms of efficiency, design replay is a successful approach. Although the implementations employed very simple adaptation and recovery strategies, all but one of the algorithms required fewer ordering heuristics than the MAID performance program. Even the simplest approach, HOBART, replayed 43.4% of the design goals.

Autonomy vs. Efficiency

Another important point supported by the empirical data is that using the performance program for recovery when the replay algorithm cannot find any more corresponding goals to replay is generally an effective strategy. Although this has been suggested by Carbonell [Carbonell, 1986] and demonstrated for non-linear planners by Kambhampati [Kambhampati, 1989a], the empirical data show that this is true if the replay strategy is sufficiently flexible.

However, there seems to be a trade-off between the amount of autonomy afforded by using the performance program for recovery, and the cost of such a strategy. The PRODIGY- and PRIAR-inspired recovery strategies represent two extremes. In general, more

Algorithm	*Goals Addressed*	*Goals Replayed*	*Heuristics Tried*	*Heuristics Used*	*% Used/Tried*
MAID	145.2	NA	1677.0	331.6	21.7
HOBART	62.5	62.5	251.9	62.5	29.8
PROBART	91.2	91.2	196.7	91.2	56.1
POSSIBLY	145.2	92.6	3928.1	361.5	14.1
PROBABLY	145.2	109.2	1390.6	158.3	28.3
BRIAR	145.2	62.5	975.0	219.7	27.1
PYRE	145.2	91.2	589.0	165.7	38.3
REMAID	145.2	111.7	584.5	276.8	59.6

Table 2: Results of running each algorithm on 20 design replay problems.

steps are reused when the history is examined every time a goal is addressed. This is the case in the POSSIBLY and PROBABLY algorithms. The problem is that it is very expensive to determine that more recovery is needed, and this usually must be done multiple times.

The other extreme is to avoid the expense of recovery altogether by giving up on the replay algorithm as soon as it can no longer suggest a goal to address. This is the approach taken by the BRIAR and PYRE algorithms. Such an approach takes advantage of the efficiency of design replay when corresponding goals are readily apparent, without spending any more effort on replay once the replay algorithm fails to find a matching goal. This technique can tolerate a larger number of goals in the new design episode that do not correspond to goals in the recorded history; however, it requires that most of the corresponding goals appear on the agenda before the replay strategy is abandoned and control is turned over to the performance program.

The REMAID Strategy

One conclusion that can be drawn from the trade-off between recovery cost and recovery autonomy is that some guide to navigating the history, other than linear search, is needed. One of the goals of the REMAID project is to find a technique for intelligently interleaving design replay and automated design. Each of the six algorithms compared above follows a strategy of aggressively pursuing only those agenda goals in the current episode that correspond to goals on the recorded history. If a single step on the history fails to find or address a corresponding goal on the agenda, it is ignored and the next step is considered. Recovery strategies are only invoked when replayable goals on the history are exhausted.

REMAID, on the other hand, attends to the kinds of failures that occur and uses that information to modify the order in which the recorded history is used so that it better matches the current agenda. Thus, it intelligently interleaves replay and automated design and produces both an efficient and autonomous design episode.

The REMAID Algorithm

The REMAID algorithm reuses the experience of the MAID automated interface design system. In addition to each goal that was addressed, and the design rule used to address it, REMAID records the heuristics that usefully reduced the size of the conflict set when the goal was selected.[6]

During replay, REMAID applies the recorded ordering heuristics to the agenda and compares the resulting candidate goals with the chosen goal recorded in the history. If only one goal is chosen from the agenda by the recorded ordering heuristics, and that goal matches[7] the chosen goal recorded on the history, then REMAID addresses that goal and continues replay with the next step recorded on the history. The assumption here is that the same goal was chosen for the same reasons, and that replay is therefore proceeding correctly. REMAID uses the PRODIGY-inspired adaptation strategy of considering alternate design rules if the recorded design rule fails.

When the recorded ordering rules do not select a matching goal on the agenda or when they select more than one goal, REMAID attempts to determine why the recorded ordering rules have failed and attempts to recover from such failures in a more appropriate way than by just going to the next step in the linear ordering of the design history. Currently, REMAID recognizes two classes of ordering rule failure: *detours* and *pretours*.

Detours If the recorded ordering rules choose more than one goal, and one of the selected goals corresponds to the chosen goal recorded on the history, then REMAID has encountered a situation where there are new goals in the current design episode that were not present in the design experience recorded in the history. Such a situation is called a *detour*. REMAID addresses the goal corresponding to the chosen goal

[6]Recall from the discussion of MAID that the number of useful heuristics is a small percentage of the number of heuristics tried.

[7]REMAID uses the same matching algorithm used by the other six algorithms.

recorded on the history first. Any other goals are assumed to be new goals which are similar enough to the chosen goal recorded on the history that they should be addressed at the same time. By recognizing detours, REMAID can use its replay strategy on goals that do not correspond to any goal recorded on the history. This strategy increases autonomy, thereby reducing the number of goals in the current episode that have to be addressed with the performance program.

Pretours If the recorded ordering heuristics choose at least one goal, and none of the goals chosen match the chosen goal recorded on the history, then REMAID assumes that there are goals in the recorded history that are not in the current design episode. Such a situation is called a *pretour*. If REMAID can find recorded goals on the history that correspond to the goals chosen in this step, then it chooses the nearest goal in the linear order of the history and continues replay there. Instead of matching each goal on the history, one at a time, against the agenda, REMAID effectively reorders the history to better fit the current agenda.

If REMAID cannot find any recorded goals on the history to match the ones chosen, or if the ordering heuristics recorded on the history for this step do not choose any goals, then it calls on MAID to select a goal. If the goal MAID selects corresponds to some goal on the history, then REMAID restarts replay with that goal, again, effectively reordering the history. Otherwise, it continues to call MAID until MAID selects a goal that corresponds to some goal on the history or the design is complete.

By recognizing pretours, and using the performance program to suggest where to restart replay, REMAID can navigate through the history, intelligently interleaving the execution of the replay and performance programs. This strategy increases autonomy without sacrificing efficiency.

REMAID Results

The REMAID algorithm was run on the same replay problems given to the other six algorithms. The averages for the 20 runs are presented in Table 2. Even though REMAID has more overhead for straightforward replay (i.e. replay with no pretours or detours), it performed as autonomously as PROBABLY and as efficiently as PYRE.

Conclusion

Most replay algorithms have been empirically tested against some performance program, but very little empirical data have been published that compare various approaches in the same domain to establish the source of power. The data presented here indicate that a flexible technique for adapting to single goal mis-matches increases both the efficiency and autonomy of replay. Further, using a performance program for recovery from replay failures is a promising technique, but there is a trade-off between efficiency and autonomy unless the replay algorithm can intelligently interleave execution of the replay program and the performance program. Although efficiency and autonomy are not the only ways that derivational analogy can be evaluated, they do give a rough comparison of the effectiveness of various approaches.

The REMAID algorithm recognizes the kinds of mismatches that occur during replay. By recognizing and addressing detours, REMAID increases the autonomy of design replay. By recognizing pretours, REMAID increases autonomy and maintains efficiency by intelligently interleaving design replay and automated design.

Acknowledgements

I am grateful to Subbarao Kambhampati, Ray Mooney, Jack Mostow, Ken Murray, Bruce Porter, and Manuela Veloso for comments, discussions and correspondence on design replay and derivational analogy. I would also like to thank Liane Acker, James Lester, and Penni Sibun for comments on early drafts of this paper.

References

[Blumenthal, 1990] Brad Blumenthal. *Replaying Episodes of a Metaphoric Application Interface Designer.* PhD thesis, University of Texas Artificial Intelligence Lab, Austin, TX, Forthcoming 1990.

[Carbonell and Veloso, 1988] Jamie G. Carbonell and Manuela Veloso. Integrating problem solving and derivational analogy. In *Proceedings of the First Workshop on Case-Based Reasoning*, 1988.

[Carbonell, 1986] Jamie G. Carbonell. Derivational analogy: A theory of reconstructive problem solving. In R. S. Michalski, Jamie G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, chapter 14. Morgan Kaufman, Los Altos, CA, 1986.

[Kambhampati, 1989a] Subbarao Kambhampati. Control of refitting during plan reuse. In *Proceedings of the International Joint Conference on Artificial Intelligence 1989*, Detroit, MI, 1989.

[Kambhampati, 1989b] Subbarao Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach.* PhD thesis, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD, 1989.

[Mostow, 1989] Jack Mostow. Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, 40(1-3), 1989.