Avoiding Pitfalls in Case-based Planning*

Jana Koehler
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3,
D-66123 Saarbrücken, Germany
e-mail: koehler@dfki.uni-sb.de

Abstract

Case-based planning is considered as a valuable tool for improving efficiency in planning by reuse and modification of existing plans. In this paper, the results of an empirical study are discussed in which several factors influencing case-based planning are investigated. The results demonstrate relative efficiency gains or losses caused by different refitting strategies, different types of plans or typical properties of the application domain and identify possible pitfalls for case-based planning.

Introduction

Recently, several complexity theoretic studies have been performed investigating worst and average cases. The comparative worst-case complexity analysis of generation and reuse under different assumptions reveals that it is not possible to prove an efficiency gain of reuse over generation (Nebel & Koehler 1993a; 1993b). The average case analysis shows that plan modification can be more efficient than plan generation under very restrictive assumptions (Bylander 1993).

Nevertheless, the two results raise the need for further investigations both theoretically and empirically. While first attempts for an empirical evaluation have already been made in the literature (Kambhampati & Hendler 1992; Hanks & Weld 1992; Veloso 1992; Bhansali & Harandi 1991), a systematic investigation of factors influencing case-based planners has not yet been performed.

In this paper, we propose to test case-based planners on the following influence factors and discuss some selected results that are obtained from an empirical analysis of the case-based planner MRL:

- **Underlying generative planning system:** Efficiency gains of plan reuse are measured relative to the effort spent on solving the same problem by planning from scratch, i.e., the efficiency of the underlying planning system influences the savings we can expect to obtain by case-based planning.
- **Similarity of planning problems:** The effort that has to be spent on plan refitting/adaptation is expected to depend on the structural similarity between the reuse candidate and the required solution. The more similar the reuse candidate is already to the desired plan, the less refitting effort is required and the better efficiency gains seem to be achievable.
- **Application domain:** Properties of the application domain can probably render reuse and modification more or less difficult leading to a different behavior of the same system in different domains. The derivation of general criteria that support the characterization of domains is a necessary prerequisite to determine the range of applicability for various case-based reasoning techniques.
- **Flexibility of refitting strategies:** Several strategies to correctly refit/adapt a reuse candidate to the desired plan have been presented in the literature. Usually, these strategies are composed of various atomic operations like *instantiation*, *deletion*, *insertion* or *reordering* of operators, but it is not known to which extent the flexibility of refitting operations influences the efficiency of case-based planning.
- **Complexity of plans**: Plan modification is supposed to become more difficult when the complexity of plans increases. One measure of complexity we analyze is the different control structures that are allowed to occur in plans, as for example *sequential composition* ($\texttt{plan}_1$; $\texttt{plan}_2$), *case analysis* (**if** `cond` **then** $\texttt{plan}_1$ **else** $\texttt{plan}_2$) and *iteration* (**while** `cond` **do** `plan`).
- **Size and structure of the plan library:** The retrieval of a good plan from a plan library is identified as a serious bottleneck for case-based planners in (Nebel & Koehler 1993b). Consequently, empirical factors influencing the efficiency of retrieval and update procedures have to be identified in order to draw conclusions for an efficient and theoretically well-founded implementation of the plan library.
- **Interaction of phases during plan reuse:** The reuse process proceeds in several phases. First, a reuse candidate has to be retrieved from the plan library, then the candidate is matched against the

*This paper has been published in the Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems, Ed. by K. Hammond, pages 104-109, AAAI Press, Menlo Park, 1994.

current planning problem and finally the candidate is refitted. Plan reuse leads to efficiency gains if

$$\text{t(retrieval)} + \text{t(matching)} + \text{t(refitting)} \leq \text{t(planning from scratch)}$$

holds. Therefore, the total effort for plan reuse including the interaction of the various phases has to be contrasted with planning from scratch.

In the following, we can only discuss some of the results and concentrate therefore on the factors

- underlying planner,
- complexity of plans,
- refitting strategy,
- application domain.

In the empirical study, we are interested in *relative* efficiency gains or losses that are caused by the influence factors.

The System MRL

MRL extends the deductive planning system PHI (Biundo, Dengler, & Koehler 1992; Bauer *et al.* 1993) with the ability to reuse and modify plans. *Plan generation* in PHI is based on constructive proofs of plan specifications in the temporal logic LLP (Biundo & Dengler 1994). LLP provides among others the modal operator $\Diamond$ (sometimes) and the binary modal operator ; (chop) which expresses the sequential composition of formulae. In using the logic LLP in a planning system it becomes possible to specify *temporary* goals with the help of nested *sometimes* operators, i.e., goals that have to be achieved at some point and not necessarily in the final state. Furthermore, universally quantified, conjunctive, and disjunctive goals and preconditions can be specified in LLP.

The example application domain of PHI is the UNIX *mail domain* where objects like *messages* and *mailboxes* are manipulated by actions like *read*, *delete*, and *save*. Furthermore, a simple blocks world taken from the SPA system (Hanks & Weld 1992) has been implemented for test purposes.

$$\begin{aligned}
&\mathsf{Plan}_{while_1} \wedge open_flag(mbox) = T \wedge \\
&\forall x\ [sender(msg(x, mbox)) = joe \\
&\qquad \rightarrow delete_flag(msg(x, mbox)) = F\,] \\
&\rightarrow \Diamond[screen_display = msgs_sender(joe, mbox) \wedge \\
&\qquad \Diamond[screen_display = all_folders(mbox) \wedge \\
&\qquad\quad \Diamond[\forall x\ [sender(msg(x, mbox)) = joe \\
&\qquad\qquad \rightarrow read_flag(msg(x, mbox)) = T \wedge \\
&\qquad\qquad\quad save_file(msg(x, mbox)) = y \wedge \\
&\qquad\qquad\quad delete_flag(msg(x, mbox)) = T\,] \wedge \\
&\qquad\qquad \Diamond open_flag(mbox) = F\,]\,]\,]
\end{aligned}$$

Figure 1: Specification of the iterative Plan *while_1*

Plans are generated by *constructively* proving plan specification formulae

$$\mathsf{Plan} \wedge pre \rightarrow goals$$

which describe the properties of the desired plan: if Plan is carried out in a situation where the preconditions *pre* hold then the *goals* will be achieved. During the proof, the planvariable Plan is replaced by a plan(formula) satisfying the formal plan specification. Figure 1 shows the specification of an iterative plan that reads, saves and deletes all messages from a sender *Joe* in the mailbox *mbox*.

Plans are represented by a certain class of LLP formulae. They may contain, e.g., basic actions which are expressed by the *execute* predicate ex, and control structures like *case analysis* and *iteration*. A plan that is generated by PHI and that satisfies the specification from Figure 1 is shown in Figure 2.

$$\begin{aligned}
&ex(from(joe, mbox))\,;ex(folders(mbox))\,; \\
&n := 1\ ; \\
&\textbf{while}\ n < length(mbox)\ \textbf{do} \\
&\textbf{if}\ sender(msg(n, mbox)) = joe \\
&\quad \textbf{then}\ ex(type(n, mbox))\,;ex(save(n, y, mbox))\,; \\
&\qquad\quad ex(delete(n, mbox)) \\
&\quad \textbf{else}\ ex(empty_action)\,; \\
&n := n + 1\ \textbf{od}\ ; \\
&ex(quit(system_mbox))
\end{aligned}$$

Figure 2: The iterative Plan *while_1*

The empirical results in this paper are obtained by testing a population of problems that is representative of the example application domain. They are illustrated with the help of the selected example plans *while_1*, *while_2*, and *if_1*. The plan *while_2* is a subplan of *while_1* only reading all messages from sender *Joe*, while the plan *if_1* is the conditional plan shown in Figure 3.

$$\begin{aligned}
&\textbf{if}\ open_flag(mbox) = T \wedge \\
&\quad delete_flag(msg(m, mbox)) = F \\
&\quad \textbf{then}\ ex(type(m, mbox))\,;ex(folders(mbox))\,; \\
&\qquad\quad ex(delete(m, mbox)) \\
&\quad \textbf{else if}\ open_flag(mbox) = T \wedge \\
&\qquad\quad delete_flag(msg(m, mbox)) = T \\
&\quad \textbf{then}\ ex(undelete(m, mbox))\,;ex(type(m, mbox))\,; \\
&\qquad\quad ex(folders(mbox))\,;ex(delete(m, mbox)) \\
&\quad \textbf{else}\ \ ex(mail(mbox))\,;ex(type(m, mbox))\,; \\
&\qquad\quad ex(folders(mbox))\,;ex(delete(m, mbox))
\end{aligned}$$

Figure 3: The plan *if_1*

Figure 4 displays the performance of the underlying generative planner PHI[1] for the generation of sequential plans containing one up to eight actions in the mail domain and the simple blocks world.

[1] PHI and MRL are implemented in SICSTUS Prolog and run on a Solbourne Sparc Station.

The efficiency of PHI results from the use of proof *tactics*, which implement a very efficient search strategy. The tactics prune the search space by guiding the planning process in a strictly goal-oriented way (Biundo & Dengler 1994).

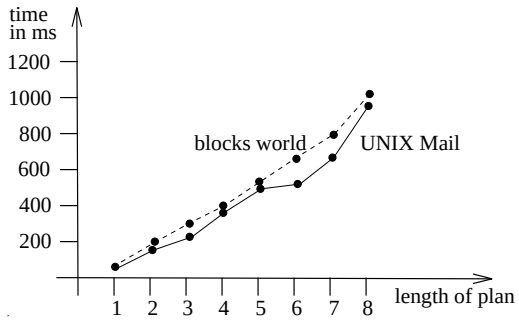


Figure 4: The underlying generative Planner PHI

Case-based planning by the MRL system is based on a logical formalization of the reuse process including the modification, representation and retrieval of plans.

Plan modification in MRL proceeds in two phases: The *matching* of the reuse candidate against the current planning problem is done deductively by proving relations between initial states and goal states. The *refitting* of the plan starts by constructing a *plan skeleton* from the reused plan depending on the result of the matching process. The plan skeleton is *extended* to a correct plan by a constructive proof of the plan specification formula which is instantiated with this skeleton (Koehler 1994c; 1994b).

The *plan library* is represented in a hybrid formalism linking the planning logic with a terminological logic. Information about plans is stored in so-called *plan entries* using the logic LLP. Each plan entry possesses an *index* which determines its position in the hierarchically structured plan library. Indices are constructed from formal plan specifications by mapping these LLP formulae to concept descriptions in a terminological logic. Thus, *retrieval* of reuse candidates from the plan library is grounded on concept classification (Koehler 1994a).

Complexity of Plans

The first experiment studies the influence of the complexity of a plan, i.e., the amount of control structures occurring in it. While existing case-based planners have been restricted to deal with sequential plans, MRL is also able to automatically reuse and modify *conditional* and *iterative* plans that are generated by the PHI planner:

- Conditional plans are generated by performing case analyses.
- Iterative plans are generated by performing inductive proofs.

MRL ensures that modified plans are correct by performing plan refitting deductively as an interleaved process of plan verification and plan generation. Plan verification is grounded on a *replay* of the proof process.

The effort necessary for plan refitting can be divided into two parts. First, the effort spent on the refitting of control structures and second, the effort spent on the refitting of the basic actions occurring in the plan.

$$\text{refitting} = \text{control structures} + \text{basic actions}$$

Table 1 shows how different control structures influence the division of the refitting effort.

Relative Refitting Effort		
Plan Type	Control Structure	Basic Action
sequential	low	high
conditional	medium	medium
iterative	high	low

Table 1: Division of Refitting Effort for Plan Types

If plans with control structures are refitted, e.g., we insert operators into a plan inside a *while*-loop, the inductive proof that has led to this *while*-loop must be replayed in order to verify the correctness of the refitted plan. The replay of a proof does only lead to marginal efficiency gains because all steps of the proof are repeated. Thus, efficiency gains during refitting come mainly from the reuse of basic actions.

Note that the time for plan modification comprises the effort spent on *plan matching*, *skeleton plan construction* as well as on *plan refitting*. This means, the savings obtained during plan refitting must at least compensate the effort spent on plan matching and skeleton plan construction in order to make plan modification more efficient than plan generation.

The modification of *sequential* plans in MRL is usually more efficient than the generation from scratch. Only minimal effort has to be spent on the verification of sequential control structures and the savings from the reuse of basic actions compensate the effort for plan matching and skeleton plan construction. This result also holds for the whole reuse process including the retrieval effort.

As for plans with *control structures* different results are obtained: The costs for the replay of proofs that verify conditional and iterative control structures mainly determine the refitting costs. The savings that are obtained by a reuse of basic actions are marginal when compared to these costs. This makes plan refitting in many cases nearly as expensive as planning from scratch. If we now consider the additional costs for matching and skeleton plan construction it becomes obvious that the effort for the modification of complex plans can be higher than the effort for planning from scratch.

In the empirical study we obtained marginal efficiency gains for the modification of conditional plans in

the mail domain when the reuse candidate is "sufficiently close" to the desired plan, i.e., for each of the cases a reusable subplan exists that achieves most of the currently required goals. The modification of iterative plans is always more expensive than planning from scratch because plan refitting is nearly as expensive as planning from scratch and furthermore, plan matching and skeleton plan construction require costly inference processes dealing with universally quantified goals and iterative control structures.

Table 2 shows typical results for the examples, where the example plan is obtained by the deletion and insertion of actions at arbitrary positions in the reuse candidate. All times in the tables are given in milliseconds.

Modification versus Generation			
Example	Generation	Modification	Relative
if_1	1590	1439	90.5 %
while_1	5810	6040	103.9 %
while_2	3690	3820	103.5 %

Table 2: Modification of Plans with Control Structures

One possibility to ensure efficiency gains for the modification of complex plans is the restriction of admissible refitting operations in such a way, that only some parts of the proofs have to be replayed in order to guarantee the correctness of the modified plan. In particular, an empirical comparison of different refitting strategies and an analysis of the trade-off between flexibility and efficiency during refitting are avenues for further research. An efficient special case of plan refitting restricted to instantiation operations is discussed in the next section.

Different Refitting Strategies

MRL implements two different refitting strategies

- *plan instantiation*
- *flexible plan refitting*

that are selected in accordance with the result of the matching process. In contrast to other case-based planners, the matching of the reused plan specification against the current plan specification is implemented as a deductive proof. The system attempts to prove that

- the reused plan is applicable in the current initial state by proving $pre_{new} \rightarrow pre_{old}$,
- the reused plan achieves at least all of the current goals by proving $goal_{old} \rightarrow goal_{new}$.

If the proofs succeed, the current plan specification $\mathsf{Plan_{new}} \wedge pre_{new} \rightarrow goal_{new}$ has been shown to be an instance of the reused plan specification $\mathsf{Plan_{old}} \wedge pre_{old} \rightarrow goal_{old}$. Therefore, an instance of $\mathsf{Plan_{old}}$ will solve the current plan specification. The instance of the reused plan is computed by extracting substitution information from the proof and applying it to the reused plan (Koehler 1994c). The instantiated plan is guaranteed to solve the current plan specification by the proof attempt. No additional verification of it must be carried out. This means, the modification costs only comprise the effort spent on plan matching and no costs for the construction of a plan skeleton and its refitting occur.

If the proof fails, refitting information is extracted from it. MRL tries to refit the reused plan by flexibly removing and adding operators, but it performs no reordering operations in order to avoid expensive computations of all possible permutations of a given operator sequence. If a operator occurs in a wrong position, it is deleted from the plan and subsequently re-introduced during the refitting process.

In contrast to the results described in the previous section where complex plans are obtained by *flexible plan refitting*, the results for *plan instantiation* show a different picture. Plan instantiation based on the proof attempt leads to remarkable efficiency gains when compared to plan generation. We show some selected results for the examples in Table 3.

Instantiation versus Generation				
Example	Generation		Instantiation	
	time	rules	time	rules
if_1	2160	205	570	97
while_1	6320	316	399	63
while_2	3800	174	200	29

Table 3: Plan Instantiation versus Plan Generation

In the experiments, a reuse candidate is supplied that can be instantiated such that it solves the current plan specification. Table 3 gives the runtime and number of deduction rules that are necessary to prove successfully the relations between preconditions and goals and contrasts it with the runtime and number of deduction rules that are necessary to generate the same example plan from scratch.

Relative Effort		
Example	time	rules
if_1	26.4 %	47.3 %
while_1	6.3 %	19.9 %
while_2	5.2 %	16.7 %

Table 4: Relative Effort for Plan Instantiation

Table 4 shows the relative effort spent on plan instantiation when the effort spent on plan generation equals 100%.

The efficiency gains of plan instantiation compared to plan generation *increase* when plans become more complex. This means, it is much easier to prove that a given complex plan can be instantiated to satisfy the current plan specification than to generate this

plan from scratch. This holds in particular for *iterative* plans. The generation of such a plan requires to perform costly inductive proofs, while the proof performed during matching is guided by a tactic running in polynomial time on the length of the formula to be proved. Thus, we obtain savings of rule applications of 80 - 85 % and runtime savings up to 95 % for iterative plans in the mail domain.

Runtime savings are usually higher than savings of deduction rules because rules applied during plan matching differ from rules applied during plan generation. The planner has to perform a *constructive* proof in order to "construct" the desired plan. It applies special-purpose generation rules that compute, e.g., appropriate instantiations of axiom schemata, which is a very costly operation. The rules applied during plan matching are less costly because they simply split a plan specification formula into its atomic subformulae and perform unification operations (Koehler 1994c).

The empirical results show that the reuse of complex plans containing control structures leads to efficiency gains when refitting operations are restricted, e.g., to instantiation. Furthermore, the advantages of a deductive approach to plan matching become apparent. The proof performed during plan matching is *sufficient* for the reuse of a plan and no additional effort has to be spent on the verification of the instantiated plan.

Different Application Domains

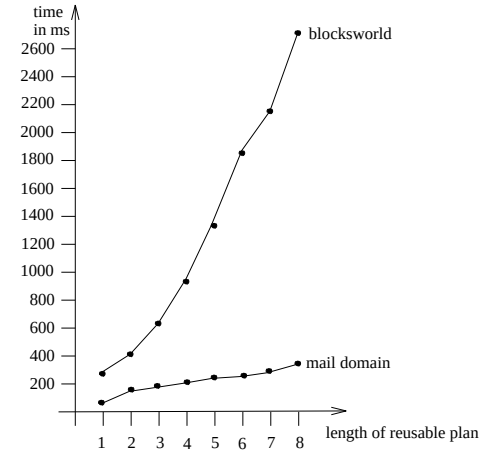
With the last experiment, we want to highlight the influence of the application domain on the performance of the system MRL by testing it on two domains:

The UNIX mail domain is an example of a *heterogeneous* application domain. This domain comprises 15 operators achieving different effects, like *open_flag(mailbox)=T*, *delete_flag(msg(x,mailbox))=F*, and *save_file(msg(x,mailbox))= file*. The objects of the domain are of various sorts like *messages*, *mailboxes* and *files*.

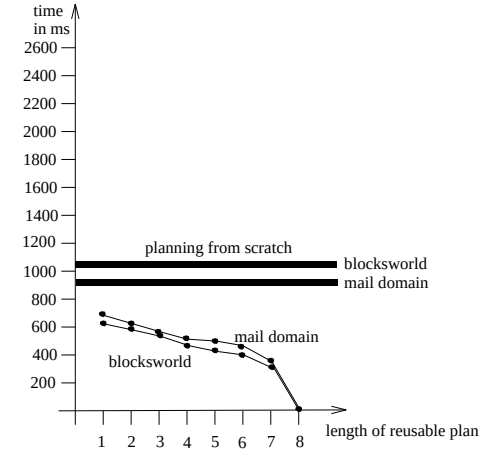
The simple blocks world taken from SPA (Hanks & Weld 1992) is an example of a *homogeneous* application domain. It comprises two operators *put_block_on_block(x,y)* and *put_block_on_table(x,table)*. Both operators achieve similar effects, namely *on(x,y)* and *on(x,table)*. As domain objects, the table and various blocks occur.

In the experiments, the same polynomial proof tactic and order-sorted unification algorithm are used for both example sets, only the domain axiomatization is exchanged. The current planning problem requires to generate a sequential plan of 8 actions by modifying a manually provided candidate plan. The candidates are chosen such that they provide an increasing starting sequence of the desired plan.

Figure 5 demonstrates that the effort for *plan refitting* is almost the same for both problems, but they differ significantly in the effort which has to be spent on *matching*. In the blocks world, matching is much more expensive because the state descriptions are homogeneous, i.e., all objects are of the same sort. This leads to many different matching possibilities. In the mail domain we have fewer objects and they are of different sorts, which makes matching less expensive since the unification algorithm can benefit from the sort information.



a) Matching



b) Refitting

Figure 5: Influence of the Application Domain

As a consequence, we can conclude that *matching* algorithms require to incorporate domain specific heuristics to a larger extend than is expected. In particular for homogeneous domains, domain dependent matching algorithms have to be developed in order to obtain an efficient runtime behavior of the case-based planning system.

Summary and Conclusion

The empirical results identify various influences on case-based planning.

One of the main results is the high influence of the underlying application domain that has not been wide-

ly discussed in the literature. In the empirical study we tested the same case-based planning system on two domains with different characteristics and distinguished *heterogeneous* and *homogeneous* domains. The result indicates that empirical studies in the blocks world are probably of a restricted value, because this domain possesses characteristics that are different from real-world applications.

A second influence factor that was studied for the first time is the complexity of plans measured on the basis of control structures occurring in plans. The treatment of control structures introduces new qualitative problems that have to be further studied. The experiments show that the results obtained for sequential plans cannot be generalized with respect to arbitrary types of plans.

Summarizing, we can conclude that case-based planners are faced with the following possible pitfalls

- The application of case-based planning techniques to application domains for which very efficient generative planning systems exist will not always lead to an efficiency gain. More research is necessary to identify the range of applicability for case-based planning in contrast to generative planning.
- Very complex planning tasks may require the generation of plans containing control structures like loops or recursion. The refitting of these plans is extremely difficult and has to be further explored. In general, the reuse of complex plans is very useful when the plan can be instantiated to the desired solution and no costly refitting operations have to be carried out.
- Homogeneous application domains make the matching process very expensive. For these domains specialized application-oriented matching algorithms have to be developed.

Being aware of these possible pitfalls helps to avoid them during the design of a case-based planning system. Design decisions taking into account properties of the application domain as well as refitting strategies and planning tasks will lead to efficient case-based planners. The generalization of the results requires further studies investigating more complex real-world domains.

Acknowledgements

I would like to thank the members of the PHI research group, Mathias Bauer, Susanne Biundo, Dietmar Dengler, Gaby Paul, and Wolfgang Wahlster. Bernhard Nebel and the anonymous referees provided helpful comments and suggestions on the draft paper.

References

Bauer, M.; Biundo, S.; Dengler, D.; Koehler, J.; and Paul, G. 1993. PHI - a logic-based tool for intelligent help systems. In *IJCAI-93*, 460–466. Morgan Kaufmann.

Bhansali, S., and Harandi, M. 1991. Synthesizing UNIX shell scripts using derivational analogy: An empirical assessment. In *AAAI-91*, 521–526. AAAI Press.

Biundo, S., and Dengler, D. 1994. The logical language for planning LLP. DFKI Research Report.

Biundo, S.; Dengler, D.; and Koehler, J. 1992. Deductive planning and plan reuse in a command language environment. In Neumann, B., ed., *ECAI-92*, 628–632. John Wiley & Sons, Chichester, New York.

Bylander, T. 1993. An average case analysis of planning. In *AAAI-93*, 480–485. AAAI Press.

Hanks, S., and Weld, D. 1992. Systematic adaptation for case-based planning. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, 96–105. Morgan Kaufmann, San Mateo.

Kambhampati, S., and Hendler, J. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55:193 – 258.

Koehler, J. 1994a. An application of terminological logics to case-based reasoning. In Doyle, J.; Sandewall, E.; and Torasso, P., eds., *KR-94*. Morgan Kaufmann, San Francisco, CA.

Koehler, J. 1994b. Correct modification of complex plans. In Cohn, A., ed., *ECAI-94*, 599–603. John Wiley & Sons, Chichester, New York.

Koehler, J. 1994c. Flexible plan reuse in a formal framework. In Bäckström, C., and Sandewall, E., eds., *Current Trends in AI Planning*, 171–184. IOS Press, Amsterdam, Washington, Tokyo.

Nebel, B., and Koehler, J. 1993a. Plan modification versus plan generation: A complexity-theoretic perspective. In *IJCAI-93*, 1436–1441. Morgan Kaufmann.

Nebel, B., and Koehler, J. 1993b. Plan reuse versus plan generation: A theoretical and empirical analysis. DFKI Research Report RR-93-33.

Veloso, M. 1992. *Learning by Analogical Reasoning in General Problem Solving.* Ph.D. Dissertation, Carnegie Mellon University.