On the Masking Effect

Milind Tambe
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pa 15213

e-mail: tambe@cs.cmu.edu

Paul S. Rosenbloom
Information Sciences Institute
& Computer Science Department
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
e-mail: rosenbloom@isi.edu

Abstract

Machine learning approaches to knowledge compilation seek to improve the performance of problem-solvers by storing solutions to previously solved problems in an efficient, generalized form. The problem-solver retrieves these learned solutions in appropriate later situations to obtain results more efficiently. However, by relying on its learned knowledge to provide a solution, the problem-solver may miss an alternative solution of higher quality — one that could have been generated using the original (non-learned) problem-solving knowledge. This phenomenon is referred to as the *masking effect* of learning.

In this paper, we examine a sequence of possible solutions for the masking effect. Each solution refines and builds on the previous one. The final solution is based on cascaded filters. When learned knowledge is retrieved, these filters alert the system about the inappropriateness of this knowledge so that the system can then derive a better alternative solution. We analyze conditions under which this solution will perform better than the others, and present experimental data supportive of the analysis. This investigation is based on a simulated robot domain called Groundworld.[1]

1. Introduction

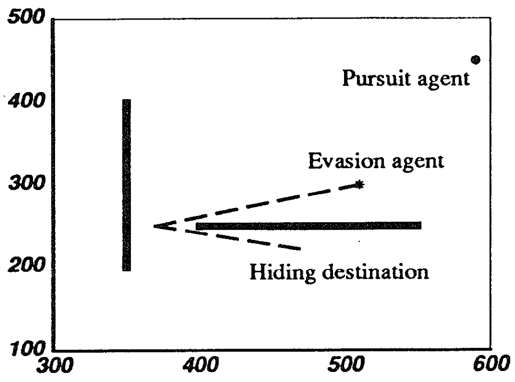
Knowledge-compilation techniques in the field of machine learning seek to improve the performance of problem-solvers/planners by utilizing their past experiences. Some examples of these knowledge-compilation techniques are explanation-based generalization (EBG/EBL) (DeJong and Mooney, 1986, Mitchell, Keller, and Kedar-Cabelli, 1986), chunking (Laird, Rosenbloom, and Newell, 1986a), production composition (Anderson, 1983, Lewis, 1978), macro-operator learning (Fikes, Hart, and Nilsson, 1972, Shell and Carbonell, 1989), and analogical and case-based reasoning (Carbonell, 1986, Hammond, 1986). These techniques store experiences from previously solved problems in an efficient, generalized form. The problem-solver then retrieves these learned experiences in appropriate later situations so as to obtain results more efficiently, and thus improve its performance.

However, by relying on its learned knowledge to provide a solution, the problem-solver may miss an alternative solution of higher quality — one that could have been generated using the original (non-learned) problem-solving knowledge. For instance, in a planning domain, the problem-solver may miss the derivation of a higher-quality plan, if a lower-quality plan has been learned earlier. The following example from the Groundworld domain (Stobie et al., 1992) illustrates this phenomenon. Groundworld is a two-dimensional, multi-agent simulation domain in which both space and time are represented as continuous quantities. The principal features in this world are walls, which block both movement and vision. Currently, our task in Groundworld involves two agents: an evasion agent and a pursuit agent. The evasion agent's task is to reach its destination from its starting point, without getting caught by the pursuit agent, and to do so as quickly as possible. The pursuit agent's task is to catch the evasion agent. Both agents have a limited range of vision. When the two agents are in visual range, the pursuit agent starts chasing, while the evasion agent attempts to escape by hiding behind some wall, from where it replans to reach its destination.

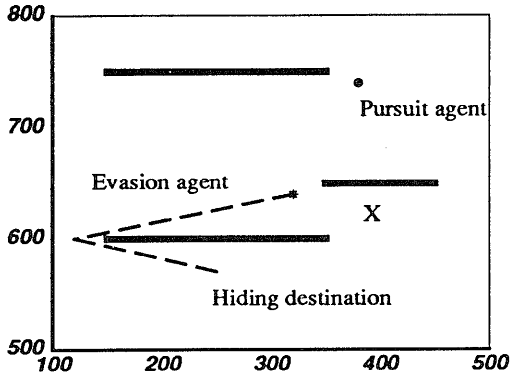
Figure 1-1-a shows part of an example scenario from Groundworld. The thick straight lines indicate walls. Here, the two agents are within visual range. To avoid capture, the evasion agent uses a map to create a plan (shown by dashed lines) to hide behind a wall. The plan is stored in learned rules, to be retrieved and reused in similar later situations. The situation in Figure 1-1-b is similar and the learned rules directly provide a plan to the hiding spot. However, by relying on these learned rules, the evasion agent misses a closer hiding spot (denoted by X). If the evasion agent had confronted the problem in Figure 1-1-b without its previously learned rules, it would have planned a path to the closer hiding spot. However, due to its learned rules, the evasion agent follows a low quality plan. While the lower-quality plan allows it to hide successfully, there is a significant delay in its hiding, which in turn delays it in reaching its real destination.

This effect, of using a low quality learned solution, has been observed for some time in humans, where it is referred to as *Einstellung* (Luchins, 1942). Modeling Einstellung in computer simulations is an important aspect of capturing human skill acquisition (Lewis, 1978). More recently, Clarke and Holte (Clark and Holte, 1992) report this effect in the context of a Prolog/EBG system, where they call it the *masking effect*, because the learned

[1]This research was supported under subcontract to Carnegie Mellon University and the University of Southern California from the University of Michigan as part of contract N00014-92-K-2015 from the Defense Advanced Research Projects Agency (DARPA) and the Naval Research Laboratory (NRL). The *Groundworld* simulator used in this paper was developed by Charles Dolan of Hughes AI center. The simulated robots in Groundworld were developed in collaboration with Iain Stobie of the University of Southern California.



(a) Learned hiding plan.



(b) The masking effect.

Figure 1-1: The masking problem when hiding: approx 15% of the Groundworld scenario is shown.

knowledge masks the original problem-solving knowledge. However, in contrast to the psychological work, Clarke and Holte's goal is not to produce this effect, but to eliminate it.

The hypothesis underlying the work described here is part way between these two perspectives; in particular, the assumption is that masking (Einstellung) is in its essence unavoidable, but that there are effective strategies that an intelligent system can use to minimize its negative consequences. Note that a low-quality solution produced due to masking is not always problematical. For instance, in real-time situations, a low-quality solution may be acceptable, as long as it is produced in bounded time (Korf, 1990). However, in other situations, a good quality solution has a much higher priority and hence avoiding the masking effect assumes importance.

We start by motivating the overall hypothesis by examining the relationship of masking to overgenerality, looking at some of the existing approaches for dealing with this overgenerality and discussing the problems these approaches have. We then propose a sequence of three new approaches to coping with masking, based on the concepts of approximations, filters, and cascades of filters. This is all then wrapped up with some analysis and backup experiments comparing these approaches.

2. Masking and Overgenerality

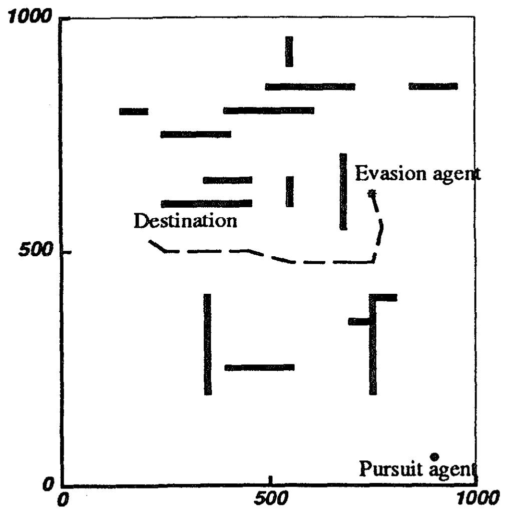
The masking effect arises because, while generating a new learned rule (i.e., at *generation time*), the system may fail to capture all of the knowledge that was relevant in deriving a high-quality solution. This may occur, for example, because the requisite knowledge is only implicit in the problem solving, or because it is intractable to capture the knowledge. Either way, the learned rule may be missing some knowledge about the exact situations where its application will lead to a high quality solution. Thus, when the learned rule is retrieved (i.e., at *retrieval time*), it may apply even though it leads to a low quality solution.

This is clearly an instance of *overgenerality* (Laird, Rosenbloom, and Newell, 1986b); however, this overgenerality is with respect to producing a high quality solution, not with respect to producing a correct solution. That is, these learned rules do not lead to a failure in task performance. For instance, in Figure 1-1-b, the learned rules that lead to masking do not result in a failure in hiding, even though a closer hiding place is missed.

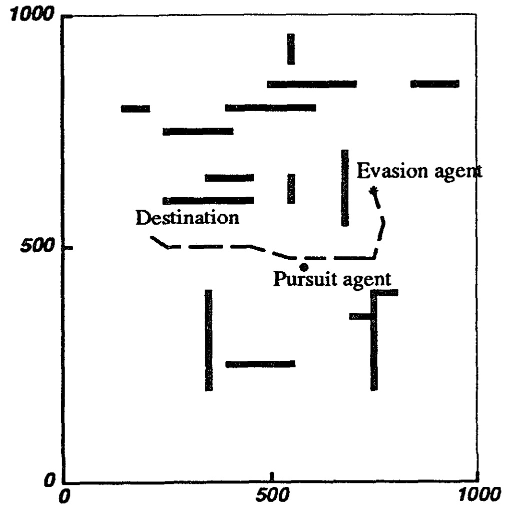
The two classical types of solutions to overgenerality are: (1) avoid it by learning correct rules, or (2) recover from it by detecting the performance failures they engender and then learning patches for those situations. Clarke and Holte's approach is of the first type. In their Prolog-based system, knowledge about solution quality is implicit in the ordering of the Prolog rules. Their EBG implementation fails to capture this knowledge while learning new rules, and leads to the masking effect. The key feature of their solution is, at generation time, to order the learned and non-learned rules according to solution quality. This ordering is guaranteed to remain valid at retrieval time, so the highest quality solution can be retrieved simply by selecting the topmost applicable rule from the ordering.

In general, solutions of type 1 — which we shall henceforth refer to as *generation-time exact* or GT-exact solutions — require capturing all of the relevant knowledge into the learned rules at generation time. However, in complex domains, it can be extraordinarily difficult to do this; that is, tractability problems result. Consider a second example from the Groundworld domain (Figure 2-1). In Figure 2-1-a, the evasion agent attempts to reach its destination, using a map to plan a path through a set of regions (Mitchell, 1988). The path (shown by a dashed line) is chosen so as to be the shortest one that avoids traveling close to the ends of walls — these are potential ambush points that may not allow the evasion agent sufficient maneuvering space to reach a hiding place before it is caught. In this particular instance, the evasion agent has no information about the pursuit agent's position, and hence cannot take that into account while planning the path; however, the pursuit agent is far enough away that it cannot intercept the evasion agent anyway.

The rule learned from the path-planning process in



(a) Learned plan.



(b) The masking effect.

Figure 2-1: Masking when trying to reach destination.

Figure 2-1-a captures a plan — a generalized sequence of regions through which the agent must traverse — that transfers to the situation in Figure 2-1-b. In this situation, the plan leads to interception by the pursuit agent. Such interceptions occur in this world, and *are by themselves a non-issue* — interceptions do not lead to failure (capture) as long as there is enough maneuvering space for successful hiding. However, in this case masking occurs because the evasion agent has knowledge about the location of the pursuit agent — from an earlier encounter with it — so it should have been possible to avoid this interception, and the resultant time lost from hiding and replanning. Without the learned rule, the evasion agent would have formed a different plan in Figure 2-1-b, one that would have avoided the area around the pursuit agent, allowing it to reach its destination quickly.

To apply the GT-exact solution to this problem, the correct learned rule would need to capture exactly the circumstances under which the path is of low quality; that is, those circumstances in which the pursuit agent is in a known location from which it can intercept the evasion agent's path. For example, the overgeneral rule could be augmented with explicit disabling conditions of the form: *(know pursuit agent in region-X)*, *(know pursuit agent in region-Y)* and so on. These disabling conditions avert the retrieval of the learned path if the pursuit agent is known to be in any of the regions from which it could intercept the path traversed.

While this approach seems plausible here, there are two problems which tend to make it intractable. First, locating all possible disabling conditions, i.e., positions of the pursuit agent for which the plan is of low-quality, involves a large amount of processing effort. This is a long path, and there are a variety of positions of the pursuit agent which threaten the path. Second, a large number of disabling conditions can severely increase the match cost of the learned rule, causing an actual slowdown with learning (Tambe, et al., 1990). The problems become even more severe in intractable domains. For example, in the chess end-game domain, it is effectively impossible to correctly condition a learned plan at generation time so as to ensure its exact retrieval (Tadepalli, 1989). As a result, at retrieval time, the learned plan may apply, but it does not always lead to a successful solution. And further in incomplete domains the relevant knowledge may not even be available at generation time. Together these problems limit the feasibility of the GT-exact approach to relatively simple domains.

The second general class of existing solutions to overgenerality are the refinement (or recovery) strategies (Gil, 1992, Huffman, Pearson and Laird, 1991, Chien, 1989, Tadepalli, 1989). However, these solutions all depend on explicit detection of failures at planning or execution time (e.g., failure in forming or executing a plan) to indicate the incorrectness of a rule, and thus to trigger the refinement process (Huffman, Pearson and Laird, 1991). While this works for overgeneral solutions that produce incorrect behavior, with the masking effect the learned solutions are only of low quality, and *do not lead to explicit failure.* Without an explicit failure, the refinement process simply cannot be invoked. (Furthermore, failure-driven learning may not always be the right strategy, e.g., in Groundworld, failure is extremely expensive — it leads to capture by the pursuit agent!) Thus, this class of solutions does not look feasible for masking problems.

3. New Approaches to Masking

The previous section ruled out refinement strategies and raised tractability issues with respect to GT-exact. This section introduces a sequence of three new approaches: (1) GT-approximate takes the obvious step of avoiding the intractability of GT-exact by approximating the disabling conditions; (2) RT-approximate improves on

GT-approximate's real-time characteristics by using the approximations as retrieval-time filters; and (3) RT-cascade refines RT-approximate by reducing the amount of replanning.

3.1. Approximating Disabling Conditions

GT-approximate overcomes the intractability issues faced by GT-exact by using overgeneral approximations (simplifying assumptions) about the exact situations for which the learned rules lead to low quality solutions. In the path-planning example, this involves replacing the set of exact disabling conditions by a single, more general, approximate condition — *(know pursuit agent's position)* — thus disabling the learned rule if any knowledge about the pursuit agent's position is available. Inclusion of only a single disabling condition also alleviates the problem of high match cost.

For this solution to be effective in general, the system must be able to derive good approximations. Fortunately, there is already considerable amount of work on this topic that could provide such approximations, e.g., (Elkan, 1990, Ellman, 1988, Feldman and Rich, 1986). However, there are still two other problems with GT-approximate. First, due to the overgeneral approximations, it may overspecialize a learned rule, disabling it from applying even in situations where it leads to a high quality solution. For instance, suppose the rule learned in 2-1-a is to be reused in 2-1-a, and *(know pursuit agent's position)* is true. In this situation, GT-approximate will disable the learned rule, even though the pursuit agent is far away, and the learned rule is thus appropriate. Second, GT-approximate does not facilitate the speed-quality tradeoff that is essential for real-time performance (Boddy and Dean, 1989). In particular, the disabling conditions used here simply disable learned rules in situations where they lead to low quality solutions, forcing the system to derive a new solution from scratch. However, in some real-time situations, a low quality response is perfectly acceptable (Korf, 1990), e.g., in the hiding situation, the evasion agent may find a low-quality plan acceptable if the pursuit agent is close and there is no time to generate a better plan.

3.2. Approximations as Retrieval-Time Filters

RT-approximate alleviates the real-time performance problem faced by GT-approximate by converting the (approximate) disabling conditions into (approximate) retrieval-time filters.[2] These filters quickly check if a learned solution is of low quality after its retrieval. For instance, *(know pursuit agent's position)* can be used as an approximate filter for the path-planning example. If this filter is true at retrieval time, then the retrieved plan is marked as being one of possibly low quality. In a time-critical situation, such as the hiding situation, the system can simply ignore this mark and use its learned solution.

Where do these filters come from? One "top-down" possibility is that they arise from explicit generation-time assumptions, much as in GT-approximate. For example, if it is known that the planning proceeded under the assumption that no knowledge is available about the location of the pursuit agent, then this assumption could be captured as a filter and associated with the learned rule. Though existence of such location knowledge at retrieval time does not necessarily mean that the plan will be of low-quality, the filter does at least ensure that the plan will not suffer from the masking effect because of this location information.

A second "data-driven" possibility is to use "significant" external events as the basis for filters. Essentially, the system notices some external object/event which may suggest to it that a retrieved solution is inappropriate. For instance, in the hiding example, if the system notices a closer, larger wall in front, then this may suggest to it that its retrieved hiding plan is inappropriate. This strategy is related to the *reference features* proposed in (Pryor and Collins, 1992), which are tags that the system associates with potentially problematical elements in its environment. Later activation of a reference feature alerts the system to a potential negative (positive) interaction of that element with its current plan.

The biggest problem with RT-approximate is that it suffers from the same overspecialization problem that dogs GT-approximate; that is, the filters are overgeneral, and can eliminate plans even when they would yield high-quality solutions.

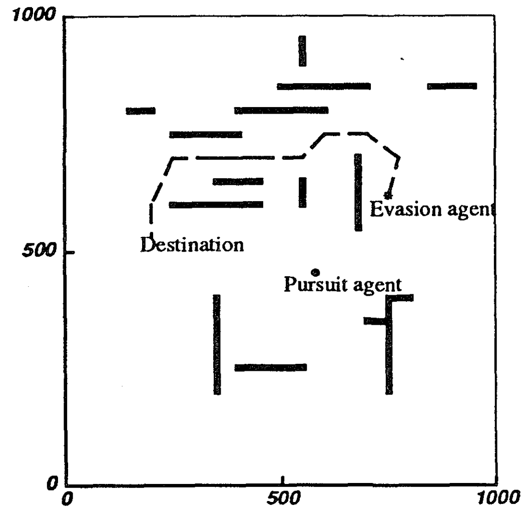
3.3. Cascading Filters

RT-cascade overcomes the overspecialization problem of RT-approximate by cascading a more exact filter after the approximate filter. It first applies the approximate filter to the retrieved solution. If this indicates that the solution may be of low quality, then the exact filter is applied to verify the solution. If the exact filter also indicates that the retrieved solution is inappropriate, then the system replans from scratch. (Sometimes, a better alternative may be to modify and re-use the existing plan (Kambhampati, 1990).) If the exact filter indicates that the solution is appropriate, then the original solution is used, thus overcoming the overspecialization introduced by the approximate filter.

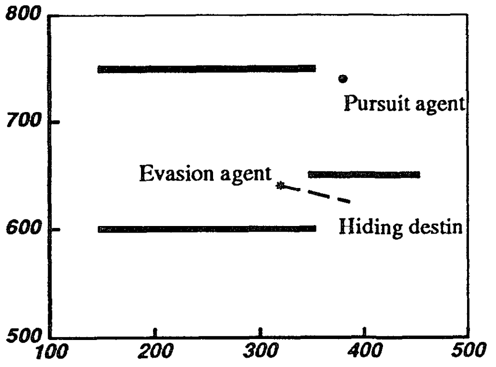
As an example, consider what happens when RT-cascade is applied to the two Groundworld scenarios introduced earlier. In the path-planning scenario, the approximate filter is *(know pursuit agent's position)* and the exact filter is a simulation of the plan that verifies whether the pursuit agent's position will lead to an interception. In the hiding scenario, the approximate filter is *(wall in front of evasion agent)*. Here, the exact filter verifies that the wall actually is a hiding place (e.g., it will not be so if the pursuit agent is located between the wall and the evasion agent), and that the wall is close. In both the scenario in Figure 2-1-b and the one in Figure 1-1-b

[2]Filtering strategies have also been used in other agent architectures. For example, in IRMA (Bratman, et al., 1988), filters decide if an external event/opportunity is compatible or incompatible with the plan the system has committed to.

the approximate filters detect possibly low quality plans. The exact filters are then run, and since they concur, replanning occurs, yielding the plans in Figure 3-1. In both of these cases RT-cascade yields the same qualitative behavior as would RT-approximate; however, in other circumstances RT-cascade would have stayed with the original plan while RT-approximate replanned. In either event, this experience can be learned so as not to repeat the exact verification (and replanning) on a similar future problem.



(a) Overcoming the masking effect in path-planning.



(b) Overcoming the masking effect in hiding.

Figure 3-1: Overcoming the masking effect.

The exact verification in RT-cascade may appear similar to GT-exact; but there is a big difference in their computational costs. In the exact verification process, the system reasons only about the single situation that exists at retrieval time. In contrast, GT-exact reasons about all possible potentially problematical situations that may arise at retrieval time. For instance, in the path-planning example, GT-exact requires reasoning about *all* possible positions of the pursuit agent that can lead to an interception, as opposed to a single position of the pursuit agent. This difference in reasoning costs at generation time and retrieval time have also been observed (and exploited) in some other systems (Huffman, Pearson and Laird, 1991). Note that this high cost of GT-exact also rules out a GT-cascade solution, which would combine the exact and approximate disabling conditions at generation time.

We have focused on applying the cascaded filters after the retrieval of a learned solution, but before its execution/application. However, the cascaded filters could be employed during or after execution as well. For instance, in the path-planning example, the cascaded filters could be invoked only if the pursuit agent actually intercepts the path. Here, this interception itself acts as a bottom-up approximate filter. The exact filter then verifies if the path is a low quality one (e.g., this path could be the best the system could plan if it had no prior knowledge about the pursuit agent, or this was the only path possible, etc.) This experience can be learned, and retrieved in future instances. However, one key problem with this strategy is that it disallows any preventive action on the problem at hand.

The key remaining question about RT-cascade is how well it performs in comparison to RT-approximate; that is, whether the extra cost of performing the exact verifications in RT-cascade is offset by the replanning effort that would otherwise be necessary in RT-approximate. This is a sufficiently complicated question to be the topic of the next section.

4. RT-approximate vs RT-cascade

Two factors determine whether RT-cascade outperforms RT-approximate. The first is the amount of overspecialization/inaccuracy in the approximate filter. Without such inaccuracy, the exact filter is simply unnecessary. The second factor relates to the cost of (re)derivation. Since the exact filter is intended to avoid the (re)derivation of a solution, it must cost less than the rederivation to generate savings. Winslett (Winslett, 1987) shows that while, in the worst case, derivation and verification processes are of the same complexity, in general, verification may be cheaper.

Let us consider two systems. The first, S-approximate, uses the RT-approximate approach; and a second, S-cascade, uses the RT-cascade approach. Now, consider a problem-instance where the approximate filter is inaccurate, i.e., it indicates that a solution is of low quality, but it is actually not of low quality. Since S-approximate depends on only the approximate filter, it will derive a new solution from scratch, and it will incur the cost of C_{derive}. On the contrary, with S-cascade, the exact filter will be used to verify the quality of the solution. This verification will succeed, indicating that the solution is actually not of low quality. Therefore, S-cascade will not derive a new solution, and will only incur the cost of C_{vsucc} for successful verification. Assuming C_{vsucc} is less than C_{derive} (as discussed above), this situation favors S-cascade. It will obtain a speedup over S-approximate of: C_{derive}/C_{vsucc}.

Thus, a cascaded filter can lead to performance improvements. However, now consider a second problem instance, where the approximate filter is accurate, i.e., it indicates that a solution is of low quality, and it is actually of low quality. S-approximate will again derive a new solution from scratch, with cost of C_{derive}. S-cascade will again use an exact filter to verify the quality of the solution. However, now since the solution is of low quality, the verification will fail, at the cost of C_{vfail}. S-cascade will then derive a new solution, at the cost of C_{derive}, so that the toal cost for S-cascade will be: $C_{derive}+C_{vfail}$. This situation favors S-approximate. It will obtain a speedup over S-cascade of: $(C_{derive}+C_{vfail})/C_{derive}$.

Thus, if the approximate filter functions inaccurately for a problem instance, S-cascade outperforms S-approximate; otherwise, S-approximate performs better. In general, there will be a mix of these two types of instances. Let N_{acc} be the number of instances where the approximate filter performs accurately, and N_{inacc} be the number of instances where it performs inaccurately. Simple algebra reveals that if S-cascade is to outperform S-approximate, then the accuracy of the approximate filter $[N_{acc}/(N_{acc}+N_{inacc})]$ must be bounded above by: $(C_{derive} - C_{vsucc})/(C_{vfail} + (C_{derive} - C_{vsucc}))$. If the approximate filter is any more accurate, S-approximate will outperform S-cascade. (It may be possible to improve this bound further for S-cascade by applying the exact filter selectively; that is, skipping it when C_{derive} is estimated to be cheaper than C_{vsucc}.)

We do not yet know of any general procedures for predicting a priori how accurate an approximate filter will be, nor how low this accuracy must be for RT-cascade to outperform RT-approximate. So, we have instead investigated this question experimentally in the Groundworld domain. Our methodology has been to implement RT-cascade as part of an evasion agent that is constructed in Soar (Rosenbloom, et al., 1991) — an integrated problem-solving and learning architecture, which uses *chunking* (a variant of EBL) to acquire rules that generalize its experience in solving problems — and then to use this implementation to gather data that lets us approximate the parameters of the upper-bound equation, at least for this domain.

Table 4-1 presents the experimental results. Since the values for C_{derive}, C_{vfail}, and C_{vsucc} vary for different start and destination points, five different sets of values were obtained. The first, second and third columns give C_{derive}, C_{vsucc} and C_{vfail} respectively (measured in number of simulation cycles). The fourth column gives the speedup — C_{derive}/C_{vsucc} — obtained by the system due to the cascaded filter, when the approximate filter is inaccurate. The fifth column gives the slowdown — $(C_{derive}+C_{vfail})/C_{derive}$ — observed by the system due to the cascaded filter, when the approximate filter is accurate. The last column in the table is the computed bound on the accuracy of the approximate filter.

Pr. No	C_{derive}	C_{vsucc}	C_{vfail}	speed cascade	slowdn cascade	Upper bound
1	340	30	11	11.3	1.03	0.96
2	276	38	11	7.3	1.04	0.95
3	234	35	6	6.7	1.02	0.97
4	176	26	8	6.7	1.04	0.95
5	83	16	--	5.2	--	--

Table 4-1: Experimental results for the path-planning example.

The first row in the table shows the data for the start and destination points as shown in Figure 2-1-a. Here, the value of 30 for C_{vsucc} represents a case where the pursuit agent is located far to the north-east of the evasion agent, so that it will not intercept the planned path. The value of 11 for C_{vfail} was obtained for the case where the pursuit agent is located as shown in Figure 2-1-b. The other four rows represent four other problems, with decreasing path lengths.

The table shows that in cases where the approximate filter is inaccurate, the system derives good speedups due to the cascaded filter. In cases where the approximate filter is accurate, the system encounters very small slowdowns due to the cascaded filter. The last column in the table shows that even if the accuracy of the approximate filter is as high as 95-97%, the cascaded filter will continue to provide the system with some performance benefits. The approximate filter that we have used — *(know pursuit agent's position)* — is not as accurate as this. For the five problems above, its actual accuracy varied from about 44% for the first problem to 28% for the last problem. We could employ an alternative filter, but its accuracy would need to be more than 95-97% before the cascaded filters become unuseful for this problem.

The last row in Table 4-1 shows a low C_{derive}, for source and destination points that are close. Here, the speedup due to the cascaded filters has decreased. However, the other entries in this last row are blank. This is because with such close source and destination points, verification failure is instant: the pursuit agent is in visual range and starts chasing. In such cases, the evasion agent abandons its path planning, and instead tries to hide.

For the hiding example, C_{derive} is 14, while C_{vsucc} and C_{vfail} are both 3. These values show little variance with different hiding destinations. This provides a bound of 73% on the accuracy of the approximate filter. If the approximate filter is any more accurate than this, the cascaded filter is not beneficial. We estimate the accuracy of our approximate filter — *(wall in front of evasion agent)* — to be approximately 25%.

5. Summary and Discussion

This paper focused on the masking problem in knowledge compilation systems. The problem arises when a system relies on its learned knowledge to provide a solution, and in this process misses a better alternative solution. In this paper, we examined a sequence of possible solutions for the masking effect. Each solution refined and built on the previous one. The final solution is based on cascaded filters. When learned knowledge is retrieved, these filters alert the system about the inappropriateness of this knowledge so that the system can then derive a better solution. We analyzed conditions under which this solution performs better than the others, and presented experimental data supportive of the analysis.

Much more needs to be understood with respect to masking. Concerns related to masking appear in different systems, including some non-learning systems. One example of this is the *the qualification problem* (Ginsberg and Smith, 1987, Lifschitz, 1987, McCarthy, 1980), which is concerned with the issue that the successful performance of an action may depend on a large number of qualifications. The *disabling conditions* for learned rules (from Section 2) are essentially a form of such qualifications. However, the solutions proposed for the qualification problem have a different emphasis — they focus on higher-level logical properties of the solutions. For instance, one well-known solution is to group together all of the qualifications for an action under a single disabling *abnormal* condition (McCarthy, 1980, Lifschitz, 1987). This condition is assumed false by default, unless it can be derived via some independent disabling rules. However, issues of focusing or limiting the reasoning involved in these disabling rules are not addressed. In contrast, our use of filters to focus the reasoning at retrieval time and the use of two filters (not just one), provide two examples of our concern with more pragmatic issues.

Masking also needs to be better situated in the overall space of learning issues. We have already seen how it is a subclass of overgeneralization that leads to a decrease in solution quality rather than outright solution failure. However, it also appears to be part of a more general family of issues that includes, among others, the utility problem in EBL.

The utility problem concerns the degradation in the speed of problem-solving with learning (Minton, 1988, Tambe, et al., 1990). Clarke and Holte (Clark and Holte, 1992) note that this is distinct from masking, which concerns degradation in solution quality with learning. However, the utility problem can be viewed from a broader perspective, as proposed in (Holder, 1992). In particular, the traditional view of the utility problem is that it involves symbol-level learning (e.g., acquisition of search-control rules), and creates a symbol-level utility problem (degradation in speed of problem-solving) (Minton, 1988). Holder (Holder, 1992) examines the problem of overfitting in inductive learning, and views that as part of a general utility problem. This overfitting problem could actually be viewed as involving knowledge-level (inductive) learning, and creating a knowledge-level utility problem (degradation of the accuracy of learned concepts). Given this perspective, we can create a 2x2 table, with the horizontal axis indicating the type of utility problem, and the vertical axis indicating the type of learning (Figure 5-1).

		Type of utility problem	
		Symbol-level	Knowledge-level
Type of learning	Symbol level	Slowdown in EBL	Masking effect
	Knowledge level	Average growth effect	Overfitting in inductive learning

Figure 5-1: A broader perspective on the utility problem.

The masking effect may now be viewed as involving symbol-level learning (knowledge compilation), but creating a knowledge-level utility problem (degradation in solution quality). Finally, the average growth effect that is observed in some systems (Tambe, 1991) provides an example of knowledge-level learning causing a symbol-level utility problem. Here, a large number of new rules acquired via knowledge-level learning can cause a symbol-level utility problem. We hope that by exploring such related issues, we can obtain a broader understanding of the masking effect.

References

Anderson, J. R. 1983. *The Architecture of Cognition.* Cambridge, Massachusetts: Harvard University Press.

Boddy, M., and Dean, T. 1989. Solving Time-Dependent Planning Problems. Proceedings of the International Joint Conference on Artificial Intelligence. pp. 979-984.

Bratman, M. E., Israel, D. J., and Pollack, M. E. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence*, Vol. *4*(4).

Carbonell, J. G. (1986). Derivational analogy: a theory of reconstructive problem-solving and expertise acquisition. In Mitchell, T.M., Carbonell, J.G., and Michalski, R.S. (Eds.), *Machine Learning: A Guide to Current Research.* Los Altos, California: Kluwer Academic Press.

Chien, S. 1989. Using and refining simplifications: explnation-based learning in intractable domains. Proceedings of the International Joint Conference on Artificial Intelligence. pp. 590-595.

Clark, P., and Holte., R. 1992. Lazy partial evaluation: an integration of explanation-based generalization and partial evaluation. Proceedings of the International conference on Machine Learning. pp. 82-91.

DeJong, G. F. and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine Learning*, *1*(2), 145-176.

Elkan, C. 1990. Incremental, approximate planning. Proceedings of the National Conference on Artificial Intelligence (AAAI). pp. 145-150.

Ellman, T. 1988. Approximate theory formation: An explanation-based approach. Proceedings of the National Conference on Artificial Intelligence. pp. 570-574.

Feldman, Y., and Rich, C. 1986. Reasoning with simplifying assumptions: a methodology and example. Proceedings of the National Conference on Artificial Intelligence (AAAI). pp. 2-7.

Fikes, R., Hart, P., and Nilsson, N. 1972. Learning and executing generalized robot plans. *Artificial Intelligence*, *3*(1), 251-288.

Gil, Y. 1992. *Acquiring Domain Knowledge for Planning by Experimentation.* Ph.D. diss., School of Computer Science, Carnegie Mellon University.

Ginsberg, M., and Smith, D. E. 1987. Reasoning about Action II: The qualification problem. Proceedings of the 1987 workshop on the frame problem in AI. pp. 259-287.

Hammond, K. 1986. Learning to anticipate and avoid planning problems through the explanation of failures. Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI). pp. 556-560.

Holder, L. 1992. The General Utility problem in EBL. Proceedings of the National Conference on Artificial Intelligence. pp. 249-254.

Huffman, S.B., Pearson, D.J. and Laird, J.E. November 1991. *Correcting Imperfect Domain Theories: A Knowledge-Level Analysis* (Tech. Rep. CSE-TR-114-91) Department of Electrical Engineering and Computer Science, University of Michigan.

Kambhampati, S. 1990. A theory of plan modification. Proceedings of the National Conference on Artificial Intelligence (AAAI). pp. 176-182.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence*, Vol. *42*(2-3).

Laird, J. E., Rosenbloom, P.S. and Newell, A. 1986. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*(1), 11-46.

Laird, J.E., Rosenbloom, P.S., and Newell, A. 1986. Overgeneralization during knowledge compilation in Soar. Proceedings of the Workshop on Knowledge Compilation. pp. 46-57.

Lewis, C. H. 1978. *Production system models of practice effects.* Ph.D. diss., University of Michigan.

Lifschitz, V. 1987. Formal theories of action. Proceedings of the 1987 workshop on the frame problem in AI. pp. 35-57.

Luchins, A. S. 1942. Mechanization in problem solving. *Psychological monographs*, Vol. *54*(6).

McCarthy, J. 1980. Circumsription -- a form of non-monontonic reasoning. *Artificial Intelligence*, *13*, 27-39.

Minton, S. 1988. Quantitative results concerning the utility of explanation-based learning. Proceedings of the National Conference on Artificial Intelligence. pp. 564-569.

Mitchell, J. S. B. 1988. An algorithmic approach to some problems in terrain navigation. *Artificial Intelligence*, *37*, 171-201.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning*, *1*(1), 47-80.

Pryor, L. and Collins, G. 1992. Reference features as guides to reasoning about opportunities. Proceedings of the Conference of the Cognitive Science Society. pp. 230-235.

Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. 1991. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, *47*(1-3), 289-325.

Shell, P. and Carbonell, J. 1989. Towards a general framework for composing disjunctive and iternative macro-operators. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence. pp. 596-602.

Stobie, I., Tambe, M., and Rosenbloom, P. 1992. Flexible integration of path-planning capabilities. Proceedings of the SPIE conference on Mobile Robots. pp. (in press).

Tadepalli, P. 1989. Lazy explanation-based learning: A solution to the intractable theory problem. Proceedings of the International Joint Conference on Artificial Intelligence. pp. 694-700.

Tambe, M. 1991. *Eliminating combinatorics from production match.* Ph.D. diss., School of Computer Science, Carnegie Mellon University.

Tambe, M., Newell, A., and Rosenbloom, P. S. 1990. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, *5*(3), 299-348.

Winslett, M. 1987. Validating generalized plans in the presence of incomplete information. Proceedings of the National Conference on Artificial Intelligence. pp. 261-266.