Incorporating Redundant Learned Rules: A Preliminary Formal Analysis of EBL*

Russell Greiner Joseph Likuski
Department of Computer Science
10 King's College Road
University of Toronto
Toronto, Ontario M5S 1A4

Abstract

"Explanation-based learning" — i.e., incorporating new redundant rules suggested by earlier problem solving experiences — is an attempt to speed up problem solving. Unfortunately, the resulting systems are not always more efficient on subsequent problems. This paper describes, analytically, whether these new rules should be added, and if so, where they should appear in the overall derivation strategy. While this task is intractable in general, we present several interesting special cases which can be solved in time (essentially) linear in the number of rules in the system.

1 Motivation

General problem solving (a.k.a. deduction) is expensive. There can be a combinatorial number of potential "solution paths" for a given query/goal — as there can be many rules/operations which each reduce the goal to a new set of subgoals, and each of these subgoals can, itself, have many possible reductions, etc. [Genesereth and Nilsson, 1987].

There are several approaches to this problem. One involves ordering the set of rules, so the first rule selected for a given (sub)goal is the one viewed as most cost effective. For example, given the *KBG* knowledge base, whose rules appear in Figures 1 and 2, we may specify that the rule R_{pm} should be used before the rule R_{pf} when determining Abe's parent — i.e., when seeking an *x* such that Parent(Abe x) holds. There can, of course, be many comprehensive "derivation strategies" — i.e., many ways of searching this knowledge base, each guaranteed to find a solution, if one exists. Our objective is to find the one which requires the least expected time. (We describe this "expected time" cost below.)

Another approach involves adding redundant information to the knowledge base, in the form of a "new" rule. Using *KBG* again, we observe that the solution to the Guardian(Abe Bart) query involved the fact Father(Abe Bart) and the rules R_{gp} and R_{pf}. This suggests modifying our derivation system for subsequent

"This research was supported by an Operating Grant from the National Science and Engineering Research Council of Canada.

R_{gp}: Parent(p q) ⇒ Guardian(p q)
R_{pf}: Father(p q) ⇒ Parent(p q)
R_{pm}: Mother(p q) ⇒ Parent(p q)

Figure 1: Rule Base associated with KB_G

queries: When asked to prove Guardian(K Y) (for any *K* and 7), this "smarter" system will immediately perform the data base retrieval of Father(k 7), and only if this fails, consider the other possible retrievals and rule-based reductions *(e.g.,* using R_{gp}, etc.). This corresponds to combining the rules R_{gp} and R_{pf} to produce the new rule

R_{gf}: Father(p q) ⇒ Guardian(p q)

which is incorporated into the set of rules, forming $KB'_G \leftarrow KB_G \cup \{R_{gf}\}$. Furthermore, this (redundant) rule is placed first, in that this system will try this new rule first in subsequent queries, before the other rules are attempted. This is the basis for the recent Explanation-Based Learning (EBL) systems [Mitchell *et* a/., 1986, Dejong and Mooney, 1986], as well as Chunking [Rosenbloom and Newell, 1982], etc.

The objective of these learning systems is *efficiency:* to improve the overall *future* performance of the system. Of course, this requires some information about these anticipated future events — especially about which questions will be posed and with what probabilities, and about the probability that certain assertions will be in the knowledge base *(KB)* when those queries occur.

Many systems implicitly employ the "obvious" assumption that "the future will mirror the past" — that the future questions will correspond to the questions asked until now. This suggests preserving *every observed rule-sequence* as a new redundant rule. Recent empirical evidence [Minton, 1988], however, has exposed some of the problems inherent in this "save all redundant rules" approach: these new rules can *slow down* the overall performance of the complete system. That is, it is not always advantageous to incorporate a proposed redundant rule into an existing knowledge base.

This paper addresses this important issue: how to decide whether to add in a new redundant rule. It assumes, as given, the *a priori* likelihood that any given database retrieval will succeed. (We may know, for example, that there is a 10% chance that the data base

retrieval "Father(κ γ)" will succeed, for any plausible pair of constants, $<\kappa, \gamma>$.[1]) It shows how to use this likelihood information to determine both whether a new rule should be added; and if so, where in the derivation strategy that rule should appear.

The next section presents the framework for this analysis. Section 3 lists our results, including both the claim that the general problem is NP-hard, and descriptions of linear time algorithms which can solve the problem for some situations. Section 4 ties this work back to EBL systems.

2 Framework

This section first provides a quick description of derivation strategies and their expected costs, in general. It then focuses on the difficulties of incorporating redundancies (read "EBL-generated rules"), within this framework.

Derivation Strategies: Given a specific query, σ, and collection of rules — like those shown in Figure 1 — we define a *derivation strategy* as an ordering which specifies when to follow which rules (to reduce the subgoal) and when to perform data base lookups. For example, one strategy for answering the query "Guardian(Abe Bart)" from KB_G would be

- Lookup Guardian(Abe Bart) from (the set of facts in) KB_G. If that succeeds, it returns "Yes" and is done. Otherwise:
- Use R_{gp} to reduce this goal to Parent(Abe Bart).
- Lookup Parent(Abe Bart) from KB_G. If that succeeds, it returns "Yes" and is done. Otherwise:
- Use R_{pf} to reduce this subgoal to Father(Abe Bart).
- Lookup Father(Abe Bart) from KB_G. If that succeeds, it returns "Yes" and is done. Otherwise:
- Use R_{pm} to reduce the Parent(Abe Bart) subgoal to Mother(Abe Bart).
- Lookup Mother(Abe Bart) from KB_G. If that succeeds, it returns "Yes"; otherwise, it returns "No". (Either way, it is now done.)

We write this strategy as $\Theta_1 = (L_g\ R_{gp}\ L_p\ R_{pf}\ L_f\ R_{pm}\ L_m)$, where R_{xy} (now) represents the reduction using the R_{xy} rule, and the L_y steps refer to lookups of the y-related propositions. We will continue to refer to R_{xy} steps as "reductions", and to L_y steps as "lookups"; collectively, these are called "steps".

We can use this same strategy, *mutatis mutandis*, to address any query of the form "Guardian(κ γ)". While this approach holds for any arbitrary κ and γ, we will focus on the situation where each is some (unspecified) constant (as opposed to an existentially quantified variable[2]).

The *expected cost* of a strategy is the (weighted) sum of the expected number of lookups plus the expected number of reductions. We assume that each lookup costs "d" cost-units, and each reduction step, "i" cost-units. Of course, the cost of following a strategy depends critically on the successes of the lookups, which in turn depend on which facts appear in the knowledge base. If all of Θ_1's lookups fail, this overall strategy will require $3i + 4d$ steps. The *expected cost* is usually less: Assuming there is a 0% chance that L_p will succeed, (*i.e.*, there are no facts of the form "Parent(κ γ)" in KB_G), and a 1%, 10% and 25% chance that L_g, L_f and L_m, respectively, will succeed,[3] then the expected cost is

$$E(\Theta_1) = \mathrm{d} + (1-.01)[\mathrm{i} + \mathrm{d} + (1-0)[\mathrm{i} + \mathrm{d} + (1-.1)[\mathrm{i} + \mathrm{d} + (1-.25)0]]] = 2.871i + 3.871d$$

There can, of course, be many strategies for a given goal within a given KB. One could, for example, not bother trying to retrieve Parent($\cdots$) from KB_G, and follow the R_{pm} rule and its associated Mother($\cdots$) lookup before R_{pf} and Father($\cdots$). The expected cost of this alternative strategy, $\Theta_2 = (L_g\ R_{gp}\ R_{pm}\ L_m\ R_{pf}\ L_f)$, is $E(\Theta_2) = d + (1 - 0.01)[i + i + d + (1 - 0.25)[i + d + (1 - 0.10)0]] = 2.7225i + 2.7325d$, which is strictly less than $E(\Theta_1)$ for any values of i and d.

Nothing forces us to consider lookups before reductions. The $\Theta_3 = (R_{gp}\ R_{pm}\ L_m\ R_{pf}\ L_f\ L_g)$ strategy, for example, does not bother to perform the (low probability) Guardian($\cdots$) lookup until the end. Its cost $E(\Theta_3) = 2.75i + 2.425d$ can be yet less expensive. If $i = 1$ and $d = 2$, then $E(\Theta_3) = 7.6 < 8.1685 = E(\Theta_2)$, meaning this Θ_3 is the best strategy yet.

This framework allows us to evaluate different strategies. As we are only considering *complete* strategies — those which are guaranteed to find an answer, if there is one — all strategies are equally likely to succeed. As such, the best strategy will be one with the least expected cost.

The number of possible strategies is *exponential* in the number of allowed steps — even for the tiny KB_G knowledge base, there are over 7! = 5,040 possible strategies, of which 42 are "depth-first".[4] [Smith, 1989], however, shows how to compute the optimal derivation strategy in a time (approximately) proportional to the number of rules, for any *disjunctive*, *irredundant* knowledge base.[5] *Disjunctive* means that all of the rules are of the form A⇒C; Smith excludes rules of the form A&B⇒C.

[1] *N.b.*, this report simply assumes that we have these probability values, and is not concerned with how they were obtained. [Likuski, 1988], as well as [Smith, 1989, Treitel, 1986], present various ways of estimating these values. *E.g.*, one method involves examining the number of assertions present in the knowledge base: as in "none are of the form Parent($\cdots$), 10% are Father($\cdots$)", etc. Of course, this also makes certain assumptions about the set of anticipated queries. There are ways of using other types of knowledge to find more accurate estimates.

[2] Here, we would seek *one* answer to a query, rather than *all* solutions. Hence, the question "Parent(Abe x)" would seek *one* parent of Abe, rather than all of his parents.

[3] We assume that these probabilities are independent.

[4] The next section defines this subset, and shows it must include an optimal strategy.

[5] That algorithm involves sorting the set of m options at each goal — hence requiring an additional factor of $O(m \log(m))$. In general, though, $m \ll$ the number of rules.

He also disallows embedded function symbols and recursive rules.[6] *Irredundant* means that there is at most one derivation path which connects any goal with any data base assertion. This means that the inference graph — the graph whose nodes are (sub)goals and whose arcs represent the rules which link a goal to its children; see Figure 2 — is a tree, rather than a more general directed acyclic graph ("dag").

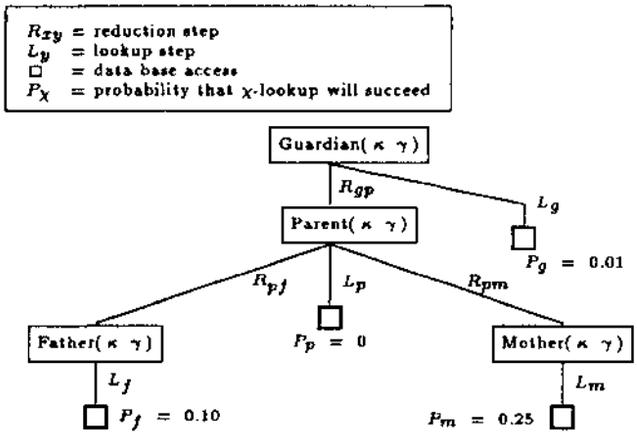

Figure 2: Inference Graph of KB_G's Rules

Dealing with Redundant Knowledge Bases: Many *KB*s, especially those extended using EBL techniques, are redundant. This paper extends Smith's results to deal with such *KB*s — *i.e.*, it describes how to find an optimal strategy for answering a given query from a *redundant* knowledge base. The above mentioned KB'_G knowledge base is redundant because its inference graph (shown in Figure 3) includes two paths which join the query `Guardian(`κ γ`)` to the lookup `Father(`κ γ`)` — one using $\langle R_{gp}\ R_{pf} \rangle$, and the other, $\langle R_{gf} \rangle$. We refer to the pair of nodes <`Guardian(`$\cdots$`)`, `Father(`$\cdots$`)`> as a "AΥ pair". We say, furthermore, that the "upward arcs" from the Υ node (here, the R_{pf} and R_{gf} ascending from `Father(`$\cdots$`)`) are *redundant* with one another,[7] and that R_{gf} is a *direct (redundant) rule* between `Father` and `Guardian`.

This paper deals only with disjunctive KBs, and assumes that each of the arguments in the antecedent or the conclusion of any rule is a variable (rather than a constant). Hence, it considers rules like `Father(p q)`$\Rightarrow$`Guardian(p q)`, but not `Father(Fred q)`$\Rightarrow$`Guardian(q Mark)`.[8]

3 Finding the Optimal Strategy in a Redundant KB

As mentioned earlier, there are an exponential number of possible strategies for a given query from any knowledge base, and *a fortiori*, from a redundant knowledge base. This section describes how to find an optimal derivation strategy in such situations, in general, and shows a linear-time algorithm which applies in some particular situations. Subsection 3.1 first addresses the task of finding an optimal strategy for a single query, and states that even this task is NP-hard in general. Subsection 3.2 then discusses (read "presents negative results for") the more general case, when there is a (known) distribution of different queries. The next section ties this analysis back to EBL systems.

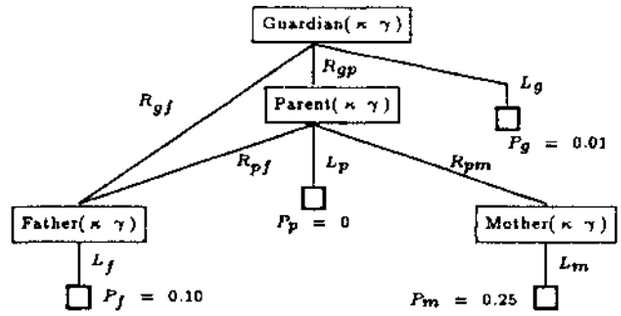

Figure 3: Inference Graph of KB'_G's Rules

3.1 Optimal Strategies for a Single Query

Use Depth-First Strategy: At any time during the derivation process, there is a *frontier* of subgoals yet to "expand", by using one or more reduction or lookup steps. A strategy could, conceivably, expand goals "below" the current frontier — *e.g.*, consider $\langle R_{gp}\ L_m\ \ldots \rangle$. We need not consider such derivation strategies, as there will always be less expensive "depth-first derivation strategies", which only expand the subgoals on this frontier. In fact, [Likuski, 1988, Lemma 3.1] proves it is sufficient to consider only strategies which consist of only "reduction chains", where each *reduction chain* is a sequence of reduction steps, which each reduce the subgoal found immediately before, followed by a lookup of the final subgoal. (*E.g.*, the sequence $\langle R_{gp}\ R_{pm}\ L_m \rangle$ is a reduction chain, as R_{pm} reduces the `Parent(`$\cdots$`)` subgoal produced by the R_{gp} step; and L_m reduces the `Mother(`$\cdots$`)` subgoal produced by this R_{pm} step. However, $\langle R_{gp}\ R_{pm}\ \underline{R_{pf}}\ L_m \rangle$ is not, as R_{pf} does *not* reduce the `Mother(`$\cdots$`)` produced by the prior step, R_{pm}.)

(The proof involves forming a depth-first strategy by "moving back" any step which interrupts a reduction chain in any non-depth-first strategy, and observing that the expected cost of the new depth-first strategy must be at least as good as the original one.)

Hence, we need only consider these "depth-first derivation strategies", and so can write any strategy, Θ, as a sequence of reduction chains, $\langle r_1\ r_2\ \cdots\ r_k \rangle$, where each r_i is a reduction chain. For example, Θ_3 can be written as $\langle\ \langle R_{gp}\ R_{pm}\ L_m \rangle,\ \langle R_{pf}\ L_f \rangle,\ \langle L_g \rangle\ \rangle$.

Use Irredundant Derivation Strategy: A derivation strategy is *redundant* if it includes the same step more than once; *e.g.*, $\Theta_4 = \langle R_{gf}\ \underline{L_f}\ R_{gp}\ R_{pm}\ L_m\ R_{pf}\ \underline{L_f}\ L_g \rangle$ is redundant as it includes L_f (*i.e.*, asks for `Father(`$\cdots$`)`) twice. It *never* makes sense to use a redundant

[6] This restricted class does include much of the relevant information in standard hierarchies, making it an important case for AI systems.

[7] We assume that our strategy-finding system knows which arcs are redundant. This is quite reasonable within the EBL context.

[8] This minor restriction is for pedagogical reasons only; [Likuski, 1988] deals with the general case.

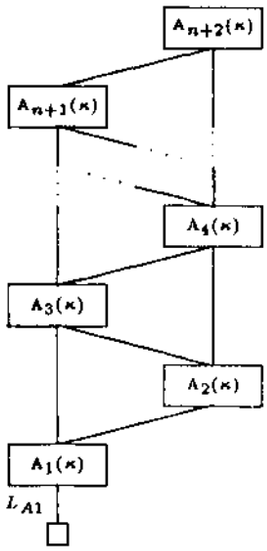

Figure 4: "Trellis Inference Graph"

derivation strategy to solve a specific query, as there is always an irredundant strategy which is functionally equivalent (*i.e.*, will find an answer whenever the redundant strategy does) and which takes strictly less time [Likuski, 1988, Lemma 3.3]. Here, $\Theta_5 = (R_{gf}\ L_f\ R_{gp}\ R_{pm}\ L_m\ L_g)$ is such a reduced, irredundant strategy for Θ_4. (Notice Θ_5 is a subsequence of Θ_4, which omits both the second L_f lookup, and the no-longer-useful R_{pf} reduction step.)

Use Irredundant Derivation Spaces: Let $RS(\Theta)$ map the strategy Θ into the set of rules it uses — e.g., $RS(\Theta_3) = \{ R_{gp}, R_{pm}, R_{pf} \}$. Notice the rule set of an irredundant strategy corresponds to an irredundant knowledge base, which means we can use [Smith, 1989]'s algorithm to find the optimal strategy in linear time. Hence, we can reduce the problem of finding the "optimal derivation strategy" to the problem of finding the "optimal derivation *space*" (where each "derivation space" is the rule set plus the needed lookup steps).

Unfortunately, there can be an exponential number of derivation spaces. Consider, for example, the "trellis" inference graph, formed from the rules

$$\{ \mathtt{A}_i(x) \Rightarrow \mathtt{A}_{i+1}(x) \}_{i=1..(n+1)} \cup \{ \mathtt{A}_i(x) \Rightarrow \mathtt{A}_{i+2}(x) \}_{i=1..n}$$

shown in Figure 4. It has $Fib(n)$ irredundant derivation spaces which connect the goal $\mathtt{A}_{n+2}(\kappa)$ to the L_{A1} lookup. ("$Fib(m)$" is m^{th} Fibanocci number.)

Fortunately, there are often ways of selecting the optimal space:

Prefer Space containing a Direct Rule: Let Θ be the optimal derivation strategy associated with the top-level goal, σ, within the irredundant knowledge base, KB; and let $(R_{\sigma 2}\ R_{23}\ \cdots\ R_{n\lambda}\ L_\lambda)$ be a reduction chain from this query to a lookup, L_λ. Now let $R_{\sigma\lambda}$ be an additional "direct rule" (not in the initial KB) which immediately connects this σ with the λ lookup. (This is the type of rule which most EBL processes would generate, after solving σ. The R_{gf} rule is an example, as it directly connects the `Guardian(`$\cdots$`)` query with the `Father(`$\cdots$`)` lookup.) Notice the set $RS(\Theta) - \{ R_{n\lambda} \} + \{ R_{\sigma\lambda} \}$ is an irredundant derivation space; let Θ' be the optimal strategy for this space. [Likuski, 1988, Lemma 3.2] proves that $E(\Theta') \leq E(\Theta)$, which means that an optimal derivation strategy (and therefore, the optimal derivation space) will include the direct rule. (This is good news for EBL fans, as these direct rules are exactly what EBL systems generate!)

Hence, there is an obvious linear time algorithm for finding an optimal derivation strategy, for the simple case of adding a new direct redundant rule to a (previously irredundant) knowledge base: Add the new direct rule, and remove the arc with which it was redundant. Then use [Smith, 1989] to produce the optimal strategy for this new KB.

A straightforward extension allows us to find, in linear time, the optimal strategy for the σ query from any "σ-direct-rule-KB" — that is, from any knowledge base which has the property that every Υ node under the goal σ includes a direct link (*i.e.*, single rule) joining it to the goal σ. (As examples: KB_G is a "`Guardian`-direct-rule-KB", vacuously, as it includes no Υ nodes; KB'_G qualifies, as its only Υ node, `Father(`$\kappa\ \gamma$`)`, connects to `Guardian`; but KB_{SG} (Figure 5) is not a "`G`-direct-rule-KB", as it includes the Υ node, `B(`κ`)`, which does not connect to `G(`κ`)`.) Here, an optimal strategy for σ can be found in the derivation space which excludes all of the non-direct links from each Υ node to σ. (The algorithm involves propagating up so-called "worth values" along these direct rules whenever possible. [Likuski, 1988, Theorem 5.1] proves that this requires examining each rule at most once.)

Task is NP-hard: Unfortunately, the general "OptDS" task — of finding the optimal search strategy in an arbitrary redundant search space — is NP-hard. The proof reduces this task to the NP-complete "Exact Covering" task [Garey and Johnson, 1979]; and appears in full in [Greiner, 1989].

3.2 Optimal Strategies for Multiple Queries

The previous subsection dealt only with a single query; in general, we may have a range of queries. This subsection assumes that we know which queries our system will be asked.[9] We would like to define a single general strategy which works *for any* anticipated query, within a forest of inference graphs.

This leads to a related, but distinct, perspective of this "worth of EBL" issue. This subsection provides four observations about these systems, which basically illustrate the additional complexity of this situation.

When *Not* To Add Direct Rules: When considering only one query, we argued above that we should always use the derivation space which includes the direct rules (whose conclusion matches that query). Unfortunately, this derivation space is less expressive than the original one, as we have to remove some rule to "make room" for

[9] In general, we may also know the frequency of these queries — e.g., that 3% of our queries will be of the form `Guardian(`$\kappa\ \gamma$`)` where κ and γ will be bound in the query, and that 8% will be of the form `Mother(`$\kappa\ x$`)` where κ is bound and x is free, etc.

the new direct one — *i.e.*, to reduce the extended KB to be irredundant. For example, we had to remove R_{pf} to accommodate R_{gf}, forming $KB_G^- = KB_G - \{R_{pf}\} + \{R_{gf}\}$. Notice this KB_G^- is not "complete", in that it can no longer use facts like `Father(Abe Bart)` to answer questions like `Parent(Abe Bart)`.

If we know that such `Parent(Abe Bart)` queries will be asked (and that facts like `Father(Abe Bart)` will be in the KB), then we cannot afford to actually change KB_G into KB_G^-. Instead, we assume there is a strategy associated with each query, which controls the derivation — here, `Guardian(` κ γ `)`'s strategy will include R_{gf} but not R_{pf}, while `Father(` κ γ `)`'s strategy will include R_{pf}. (This is easy to implement using a system like MRS [Russell, 1985].)

We might consider a heuristic of performing such KB-modifications (*e.g.*, augmenting KB_G with R_{gf}) only if this addition leads to a real benefit — *i.e.*, if the optimal strategy with the new rule is *strictly* better than the original optimal strategy, for the given query. Unfortunately, this approach can miss really good strategies.

Consider asking the queries `Guardian(Abe Bart)` and `Guardian(Abe Cici)` from the KB_G knowledge base, which includes `Father(Abe Bart)` and `Mother(Abe Cici)`. We begin with the strategy which is optimal for our values of i and d, $\Theta_3 = \langle R_{gp}\ R_{pm}\ L_m\ R_{pf}\ L_f\ L_g \rangle$.

After the first query, the EBL system would propose the direct rule R_{gf}, and suggest adding it to KB_G. Notice the optimal `Guardian` strategy, in this derivation space, is $\Theta_6 = \langle R_{gp}\ R_{pm}\ L_m\ R_{gf}\ L_f\ L_g \rangle$, whose expected cost matches $E(\Theta_3)$. Hence, the above heuristic would veto this rule, and tell us to keep the original KB_G, and continue using the original Θ_3 strategy.

After solving the second query, this EBL system would propose the rule

$$R_{gm}\text{: } \texttt{Mother(p q)} \Rightarrow \texttt{Guardian(p q)}$$

and suggest adding it to KB_G. As the optimal strategy using this new rule — $\Theta_7 = \langle R_{gm}\ L_m\ R_{gp}\ R_{pf}\ L_f\ L_g \rangle$ — is better than Θ_3 (as $E(\Theta_7) = 2.5i + 2.425d < E(\Theta_3)$), we would then add this R_{gm} to KB_G.

Notice we are still missing the big win, which involves *both* new direct rules. Here, the optimal strategy would be $\Theta_8 = \langle R_{gm}\ L_m\ R_{gf}\ L_f\ L_g \rangle$, with expected cost $E(\Theta_8) = 1.75i + 2.425d$, which is much less than $E(\Theta_7)$ as it no longer even considers the `Parent(`$\cdots$`)` subgoal!

Use a Fixed Ordering of Steps? Consider an inference graph which has the goal `G(`κ`)` leading down to the (inferior) subgoal `S(`κ`)`, and let Θ_S (resp., Θ_G) be the optimal strategy for `S` (resp., `G`). It would be convenient if Θ_G necessarily "includes" Θ_S — *i.e.*, if each of Θ_S steps appear in Θ_G, and they all appear in the same order as they occur in Θ_S.

This would mean, in particular, that after finding the optimal strategy for the "root" of an inference "tree", one could re-use that ordering on every node in that tree. For example, we could then "subset" the optimal strategy for the `Guardian(` κ γ `)` query to form the optimal strategy for `Parent(` κ γ `)` queries: while that strategy would include only R_{pf} and R_{pm} and the relevant lookups (but not R_{gp}), it would include these steps in the same order here as they appeared within `Guardian(` κ γ `)`'s strategy.

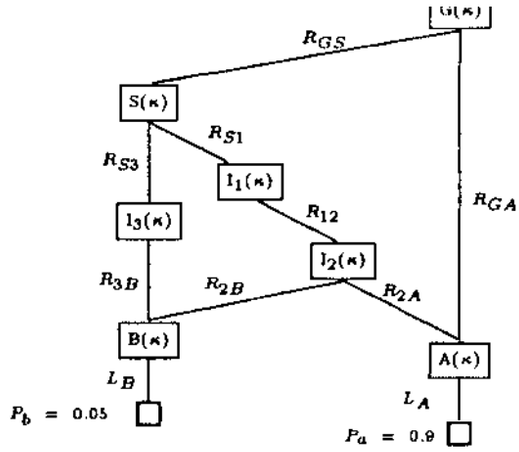

Figure 5: Inference Graphs for `S(x)` and `G(x)` — KB_{SG}

This would greatly simplify the task of finding the optimal derivation strategies for a set of queries from a given KB: one need only find the strategy for each root of the "inference forest". It would also mean that a PROLOG system could achieve optimal performance by simply asserting its rules and data base facts in this order.

Fortunately, this "fixed ordering" property holds for any irredundant knowledge base; see [Likuski, 1988, Lemma 4.5]. Unfortunately, it does *not* hold for redundant knowledge bases; see [Likuski, 1988, Lemma 4.6].

Use a Fixed Set of Rules? The situation is really even worse: not only might we have to find a different ordering on the steps as we go from one goal to its "superior" goal, we might, in fact, have to use a completely different derivation space! That is, there can be steps within Θ_S which do not even appear anywhere within Θ_G.

Figure 5 is an example. The optimal strategy for `S(`κ`)` is $\Theta_S = \langle R_{S1}\ R_{12}\ R_{2A}\ L_A\ R_{2B}\ L_B \rangle$. Notice, however, the optimal strategy for `G(`κ`)` is $\Theta_G = \langle R_{GA}\ L_A\ R_{GS}\ R_{S3}\ R_{3B}\ L_B \rangle$, which does not include *any* of Θ_S's reduction steps!

When are Efficient Algorithms Possible? While the general OptDS problem is NP-hard, we presented efficient (in fact, linear time) algorithms which work in some specific situations. This final portion suggests a way of characterizing which situations may be tractable.[10]

This fixed-ordering property is essential to [Smith, 1989]'s algorithm for finding optimal strategies. That algorithm works in a "bottom-up" fashion: propagating so-called "worth values" up from lookups, via reduction steps. It works in linear time because the decision of which step to use, at a given subgoal, depends only on these steps and their "children". As this property also holds in the "direct-rule" case, we could use (a trivially modified version of) Smith's algorithm here as well.

[10] We assume, of course, that $P \neq NP$ — *i.e.*, that all NP-hard problems are intractable.

Unfortunately, the above points show that redundant inference graphs, in general, need not have this nice property. That is, an optimal strategy found for an inferior S(k) node may be completely irrelevant, when seeking the optimal strategy for its superior G(k) node, meaning our algorithm would have backtrack to consider other possibilities, as it works its way up towards the top goal. We feel that this back-tracking is what leads to the intractability of the general task, which is why we expect there are efficient algorithms for finding optimal strategies *only* in situations where the decisions made at one node are "honored" at all superior nodes.

4 Conclusions

This concluding section first ties this work back into the framework of EBL systems in general, and then lists some obvious extensions to this work.

Tie to EBL Systems: The positive result mentioned above — that a direct redundant rule (i.e., the result of an EBL system) can never slow down a derivation system — should be viewed as only a partial vindication of EBL systems and techniques. Below are four comments about this claim:

- Most EBL systems leave in both the direct rule, and the rules from which it was derived — *e.g.,* both the derived R_{gf} and the pair R_{gp} and R_{pf}. We showed that this is never efficient, even for the query itself.
- Most EBL systems move this new rule to the *beginning* of the system's derivation strategy; this is not always the optimal place. (Recall that we added R_{gf} towards the back of the 06 strategy. In fact, the expected cost of the strategy which includes R_{gf} in the front of the strategy is strictly *worse* than the original, $\backslash ne\text{-}R_{gf}$ strategy, 0_3!)
- Section 1 mentioned two ways of improving the expected cost of a derivation — (1) by determining the best strategy, and (2) by adding redundancies. As empirical evidence has shown that using (2) without (1) can produce arbitrarily inefficient systems, this report has examined ways of combining both of these.
- This result applies only when the prior knowledge base is irredundant, and it only deals with a single query. As shown above, the situation is much more complicated when we consider multiple questions, and arbitrarily redundant knowledge bases.

Extensions: We have only scratched the surface of this analysis; there are many other areas to consider as well. The first obvious arena is handling conjunctive and recursive knowledge bases. Another is to combine this approach with other control strategy mechanisms — including conjunct ordering [Smith and Genesereth, 1985] and forward chaining [Treitel and Genesereth, 1987]. The third is to obtain more accurate empirical values. For example, we have assumed that the costs of reductions and lookups (read "i" and "d'') are uniform. Preliminary empirical observations show that these costs depend on the number of variables, etc.

Results: This report takes seriously the view that Explanation-Based Learning is a method for improving the future performance of a reasoning system. This leads to the formal foundation for analysis presented in Section 2, which is based on the expected cost for solving certain queries from a given knowledge base (based on a given distribution of facts). Section 3 uses this framework to describe both the complexities (read "NP-hardness") inherent in this undertaking; and certain restricted situations where efficient algorithms (based on [Smith, 1989]'s work) are possible. Section 4 uses this framework to understand why EBL systems do, and do not, succeed in their attempts to improve the performance of their underlying systems.

References

[DeJong and Mooney, 1986] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning,* l(2):145-76, 1986.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. II. Freeman and Company, New York, 1979.

[Genesereth and Nilsson, 1987] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[Greiner, 1989] Russell Greiner. Incorporating redundant learned rules: A preliminary formal analysis of EBL. Technical report, University of Toronto, 1989.

[Likuski, 1988] Joseph Likuski. Integrating redundant learned rules in a knowledge base. Master's thesis, University of Toronto, October 1988.

[Minton, 1988] Steven Minton. Quantitative results concerning the utility of explanation-based learning. In *AAAI-S8,* pages 564-69, San Mateo, CA, August 1988. Morgan Kaufmann Publishers, Inc.

[Mitchell *et ai,* 1986] Thomas M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Example-based generalization: A unifying view. *Machine Learning,* 1(1):47-80, 1986.

[Rosenbloom and Newell, 1982] Paul S. Rosenbloom and Allan Newell. Learning by chunking: Summary of a task and a model. In AAA/-&2,*Pittsburgh, August 1982.

[Russell, 1985] Stuart Russell. *The Compleat Guide to MRS,* June 1985. Stanford KSL Report HPP-85-12.

[Smith and Genesereth, 1985] David E. Smith and Micheal R. Genesereth. Ordering conjunctive queries. *Artificial Intelligence: An International Journal,* 26(2):171-215, May 1985.

[Smith, 1989] David E. Smith. Controlling backward inference. *Artificial Intelligence: An International Journal,* 39(1), 1989. (Also Stanford Technical Report LOGIC-86-68).

[Treitel and Genesereth, 1987] Richard J. Treitel and Michael R. Genesereth. Choosing orders for rules. *Journal of Automated Reasoning,* 3(4):395-432, December 1987.

[Treitel, 1986] Richard J. Treitel. *Sequentialization of Logic Programs.* PhD thesis, Stanford University, November 1986. Technical Report STAN-CS-86-1135.