Is There Any Need for Domain-Dependent Control Information? A Reply

Steven Minton
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
minton@isi.edu

Abstract

In this paper, we consider the role that domain-dependent control knowledge plays in problem solving systems. Ginsberg and Geddis (Ginsberg & Geddis 1991) have claimed that domain-dependent control information has no place in declarative systems; instead, they say, such information should be derived from declarative facts about the domain plus domain-independent principles. We dispute their conclusion, arguing that it is impractical to generate control knowledge solely on the basis of logical derivations. We propose that simplifying abstractions are crucial for deriving control knowledge, and, as a result, empirical utility evaluation of the resulting rules will frequently be necessary to validate the utility of derived control knowledge. We illustrate our arguments with examples from two implemented systems.

Introduction

In a AAAI paper entitled "Is there any Need for Domain-Dependent Control Information?" Ginsberg and Geddis (1991) (henceforth G&G) consider whether domain-dependent control knowledge is necessary for domain-independent problem solving systems. Their conclusion is aptly summarized by their abstract, which consists of the single word: "No". They argue that all domain-dependent control information can be, and should be, derived from simple declarative facts about the domain plus domain-independent control knowledge. In this paper we reconsider the question raised by G&G and arrive at a different conclusion. We argue that in many cases, domain-dependent control information cannot, in a practical sense, be derived solely in the manner specified by G&G, due to the complexity of the reasoning that would be required. In such cases empirical testing will be required to validate the utility of control knowledge. Consequently, one cannot dispense with the control knowledge as simply as they imply.

The contribution of this paper is not simply to rebut the claim made by G&G, but more importantly, to investigate reasoning strategies for producing domain specific control knowledge. In this vein, we propose that, due to the complexity required to derive control rules, *simplifying abstractions* and *empirical utility evaluation* can be valuable tools.

We begin this paper by reviewing G&G's arguments. Despite our disagreement with their conclusion, we believe the issues they raise are important, and their basic observations have considerable merit. Nevertheless, they leave the reader with an over-simplified view of the world. In order to show this, we consider some concrete examples that illustrate the complexities involved in acquiring control knowledge. We begin by reviewing the learning process used by PRODIGY-EBL (Minton 1988), and then turn our attention to some recent experiments with MULTI-TAC (Minton 1996).

Ginsberg and Geddis' Argument

Ginsberg and Geddis (G&G) begin by introducing the following distinction. They propose that meta-level information consists of (at least) two different types: meta-level information about one's base-level knowledge, which they call modal knowledge, and meta-level information about what to *do* with the base-level knowledge, which they call control knowledge. An example of a modal sentence is: "I know that Iraq invaded Kuwait". An example of a control rule is: "To show that a country c is aggressive, first try showing that c invaded another country".

Meta-level information can be either domain-dependent or domain-independent. For instance, the control rule we just mentioned is domain-dependent. An example of a domain independent rule is: "When attempting to prove that a goal is true, choose a rule with the fewest subgoals first".

The crux of G&G's argument that domain-dependent control knowledge is unnecessary rests on a simple, but important, observation: the behavior of any domain-independent problem solving algorithm depends only upon the syntactic form of the base-level domain theory. They present the following Prolog theory as an example:

```
hostile(c) := allied(d, c), hostile(d).
hostile(c): = invades-neighbor(c).
```

```
allied(d,c):= allied(c,d).
invades-neighbor(Iraq).
allied(Iraq, Jordan).
```

To prove `hostile(Jordon)`, one must eventually expand the subgoal `Invades-neighbor(Iraq)`, rather than pursuing the infinite chain of subgoals:

$$hostile(Jordan), hostile(Iraq), hostile(Jordan)\ldots$$

Thus, one appropriate control rule for this domain is that the subgoals involving Invades-neighbor should be tried before the subgoals involving hostile. This might be expressed as follows:

```
Delay(hostile)
```

However, as G&G point out, the rationale for this control rule is domain independent. If one were to take our prolog theory and change the predicate names and constant names, the same control rule would still hold (modulo the fact that the predicate names would have to be changed in the rule as well), as illustrated below:

```
acute(c) := congruent(d, c), acute(d).
acute(c): = equilateral(c).
congruent(d,c):= congruent(c,d).
equilateral(T1).
congruent(T1, T2).
```

Here, a similar control rule, this time involving acute vs. equilateral, is valid:

```
Delay(equilateral)
```

The intuition that control knowledge only depends on the form of the base-level theory is the basis for G&G's claims. Explaining their view, they quote David Smith's thesis (Smith 1985):

> Good control decisions are not arbitrary; there is always a reason why they work. Once these reasons are uncovered and recorded, specific control decisions will follow logically from the domain-independent rationale, and simple facts about the domain.

G&G's paper is divided into two parts. In the first part, they prove that any control rule can be replaced by a domain independent control rule and a modal sentence describing the structure of the search space (with only a minimal effect on efficiency). For example, for the "hostility theory" above, G&G abbreviate the required modal sentence using a "type" predicate, which captures the relevant structural aspects of the theory:

$$Type_{37}(hostile, invades{-}neighbor, allied, Iraq, Jordan)$$

Since the "triangle theory" has the same structure it would also be a $type_{37}$ theory:

$$Type_{37}(acute, equilateral, congruent, T_1, T_2)$$

Once the requisite modal sentence is formalized, the domain independent form of the control rule is simple to express, e.g.:

$$Type_{37}(p_1, p_2, p_3, o_1, o_2) \rightarrow delay(p_1)$$

This rule indicates that for $Type_{37}$ Prolog theories, with arbitrary predicate symbols p_1, p_2, p_3 and constant symbols o_1 ,o_2, subgoals involving predicate p_1 should be delayed.

As G&G admit, this construction "is in many ways trivial, since in order to transfer a control rule from one domain to another we need to know that the two search spaces are identical". Nevertheless, the intuition underlying the construction is more general, since, as G&G explain, the "arguments underlying domain-dependent control rules do not typically depend on the complete structure of the search space". Therefore, in the second part of their paper, they go on to examine several examples of domain-dependent control rules and to informally elaborate the rationales underlying these rules. For instance, they consider planning a trip from Stanford to MIT. The domain-dependent control knowledge is this: "When planning a long trip, plan the airplane component first". The rationale for this control rule is based on two domain-dependent observations. First, the three legs of the journey (one in the air and two on the ground) are non-interfering, except for scheduling concerns, and second, the subproblem of selecting air transportation is likely to be more constrained than arranging for ground transportation. G&G point out that, given these two facts about the domain, the afore-mentioned control rule can be inferred in a domain-independent fashion.

Our main quarrel with G&G is that they leave the reader with an over-simplified view of the world. In their introduction they state: "The claim we are making - and we intend to prove it - is that there is no place in declarative systems for domain-dependent control knowledge". In fact, the only formal proof in the paper involves the construction in part 1 which trivially divides every control rule into a modal fact and a domain-independent rule, accomplishing little more than a change in terminology. The examples analyzed in the second part of their paper are more interesting, but at the end of this section they state:

> Although the control rules used by declarative systems might in principle be so specific as to apply to only a single domain, the domain-independent control rules in *practice* appear to be general enough to be useful across a wide variety of problems... Whether or not this observation is valid is more an experimental question than a theoretical question...

In this regard their work is really a proposal (not a proof) and the practical significance is left for us to address in this paper.

If we carefully consider their claim that domain-dependent control rules can (and should!) always be derived from simple facts about the domain, a number of questions arise. Are the required modal facts

- the facts about the domain - always simple? Is domain-independent inference always sufficient? Will the derivation process be efficient? These questions are not considered by G&G, but as we will see, they can be of critical importance.

We will henceforth restrict our consideration to the practical application of G&G's approach, as presented in the second part of their article. Thus, we assume that the domain-independent principles that G&G would use to derive domain-dependent control knowledge are substantial and broadly applicable, unlike the trivial transformation discussed above.

Control Knowledge in Prodigy-EBL

Prodigy's Explanation-based Learning component (Minton 1988; Minton *et al.* 1989) acquires domain-dependent search control rules by analyzing problem-solving traces. Commenting on PRODIGY-EBL, G&G propose that instead of acquiring new control rules, it would be preferable to acquire new base-level or modal information. In fact, this criticism is largely misguided, since, as we will explain, PRODIGY-EBL actually does learn domain-dependent modal information and then converts this information into search control rules, just as G&G recommend. Thus, in many respects, PRODIGY-EBL supports G&G's contention that control knowledge can be acquired by learning domain-dependent modal information, and then converting that information into domain-dependent control rules using domain-independent principles. However, as we will see, there are some crucial pieces missing from G&G's story.

Simplifying Abstractions in Prodigy-EBL

Let us examine PRODIGY-EBL's learning process in more detail. After the Prodigy problem solver finishes solving a problem (or subproblem), PRODIGY-EBL examines the trace, looking for instances of its target concepts. For example, one target concept is GOAL-INTERACTION. A goal interaction occurs when achieving one goal necessarily clobbers (i.e,. deletes) another goal. Consider, for instance, a blocksworld problem in which both (On A B) and (On B C) are goals. If the solver achieves (On A B) before (On B C), then all ways of subsequently achieving (On B C) will clobber (On A B). A solution is still possible, since (On A B) can be re-established after block B is put on top of block C, but this solution is suboptimal. By constructing an explanation (a proof) as to why the interaction occurred, and finding the weakest conditions under which the proof holds, PRODIGY-EBLis able to generalize from the example, learning that achieving (On x y) before (On y z) will result in a goal interaction. Notice that this is a modal fact, in the sense of G&G.

This modal fact is then converted into a control rule (a goal ordering rule) which says that if both (On x y) and (On y z) are goals, then (On y z) should be solved first.[1] The domain-independent justification for this last step is actually rather complex. In general, it is preferable to avoid clobberings because they tend to decrease the probability of finding a solution and, in addition, when solutions are found, the finished plans tend to be longer (since clobbered goals have to be re-established). Prodigy does not explicitly carry out any of this reasoning – it is, in effect, hardcoded into the system. The concept of a goal-interaction provides a simplifying abstraction, so that we can ignore the complexities mentioned above, i.e., we *assume* that avoiding goal interactions is preferable. Avoiding goal interactions isn't *necessarily* preferable (as discussed by Minton, 1988). For instance, it may turn out that all paths leading to a solution, or the shortest paths to the solution, involve one or more goal-interactions.

G&G propose that, given domain-dependent modal knowledge, one can infer the necessary domain-dependent control knowledge, but they do not investigate in any depth the nature of the reasoning process. In particular, they do not discuss any role for simplifying abstractions in the reasoning process. We use the term "abstraction" (albeit informally) because the essential characteristic of a simplifying abstraction is that it deals only with a restricted aspect of the domain or the search space. We use the term "simplifying" because the theory is often incomplete, or involves some other simplifying assumptions, so that worst-case complexities are avoided. We claim that simplifying abstractions are often necssary in the reasoning process. However, we also recognize that control decisions based on such abstractions will not always be optimal, because of the simplifications involved (e.g., the simplifying assumptions may be violated). For example, the control rule "Achieve (On y z) before (On x y)" that was learned in our blocksworld example is not guaranteed to recommend the best course of action. As we will explain later, PRODIGY-EBL is able to recover from this (to some extent) by empirically evaluating the utility of the learned control rules and to "discard" any rules with negative utility, but G&G's scheme does not allow for this option. In G&G's scheme, one would only have the modal fact "Achieving (On x y) before (On y z) results in a goal interaction", and the domain-independent control rule "Avoid goal interactions". Since there is no place in their scheme for domain-dependent control rules, they cannot determine the utility of such rules and discard them when appropriate. In fact, they do not discuss

[1] In an aside, G&G question why the rule learned by PRODIGY-EBL on this example states that (On y z) should be *solved* before (On x y). They point out that conceptually, the execution order is not necessarily the same as the planning order. The answer is that in Prodigy, the order in which goals are solved *does* determine their ordering in the plan, because Prodigy is a means-ends, total-order planner. So there is no distinction made between "planning order" and "execution order".

the possibility that their reasoning process may produce counter-productive results.

Simplifying abstractions play a crucial role in PRODIGY-EBL. Our blocksworld example illustrated one type of simplifying abstraction, which is used when learning goal-ordering rules. Another type of simplifying abstraction underlies the entire EBL approach. Consider that, in general, the utility of a control rule depends on all the following factors:

- the cost of matching the rule, and
- the frequency which with rule is applicable, and
- the benefit derived when the rule is applied.

The explanation process considers only the structure of the search space, as opposed to the quantitative costs/benefits of the decision process. Thus the first two considerations are ignored when proposing a control rule (and the third is only partially dealt with). This constitutes a simplifying abstraction, above and beyond the simplifying abstractions used within the explanation process. We know of no simple way to reason about these factors in order to derive high-utility control rules.

In order for G&G's scheme to be successful, one must not only be able to derive control information from the modal facts, but the process must be accomplished *efficiently*. In this section we have argued that simplifying abstractions can be essential for making the reasoning process efficient.[2] Many varieties of simplifying abstractions can be found in analytic learning systems that derive control knowledge.[3] Some examples include: the "preservability" assumption used by Bhatnagar and Mostow's FAILSAFE (Bhatnagar & Mostow 1994), the focus on non-recursive explanations in Etzioni's STATIC (Etzioni 1993), the "plausible" domain theory used to identify chronic resource bottlenecks in Eskey and Zweben's constraint-based payload scheduler (Eskey & Zweben 1990), and the use of depth-limits, domain-axioms and np-conditions by Katukam and Kambhampati (1994).

The Value of Utility Evaluation

The control rules generated by PRODIGY-EBL are not guaranteed to be useful, because, as we have seen, they are based on simplifying assumptions. In fact, most rules typically have negative utility! As a result, PRODIGY-EBL empirically evaluates each rule's utility by measuring its costs and benefits during subsequent problem solving episodes. Thus, in the blocksworld, a control rule that orders the goal (ON y z) before the goal (ON x y) will be probably be kept, since it is likely to improve the solver's performance. In contrast a rule that orders (Clear x) after (On x y) will have low benefit in most cases (even though it is appropriate and has low match cost), because achieving these goals in the wrong order will not generally decrease efficiency. (Note that (Clear x) is both a precondition and postcondition of achieving (On x y).)

In general, finding a good set of control rules is a complex problem, even when empirical utility evaluation is used; one complicating factor is that utility of an individual control rule is influenced by the choice of which other control rules to include in the system. For instance, the rule mentioned above that orders (Clear x) after (On x y) can, perhaps surprisingly, be very useful in some circumstances, because if there are blocks on top of x, and they are mentioned in other ON goals, it can be preferable to delay (Clear x), until all the ON goals have been achieved. However, this is only necessary if, while achieving (Clear x), the system is likely to do some action that interferes with one of these other goals. Thus, the utility of this rule depends on the other rules in the system. (Minton (1988) and Gratch (1995) describe schemes for evaluating the utility of control rules.)

It is difficult to imagine that any domain-independent system could logically derive guaranteed, high-utility control rules for complex domains, without any empirical component (and we are unaware of the existence of any such system).[4] However, this is what G&G are implicitly suggesting when they recommend that the control rules themselves can be dispensed with. They do note that probabilistic domain-dependent modal information may play a part in the derivation, but the problem, of course, is knowing what probabilistic information will be sufficient to guarantee the utility of a control rule. Thus, we claim that the rationale for a control rule, i.e., the derivation based on domain-independent principles and domain-dependent modal information, will in many cases be incomplete. (Note that we are not claiming that a complete rationale does not exist, but simply that for practical reasons, the rationale generated by a system will often be based on simplifying assumptions.)

For this reason, we dispute G&G's claim that control knowledge has no place in declarative systems. We need to explicitly represent the rules so we can test them and so that we can indicate which rules should actually be used for problem solving.

[2] The simplifying abstractions that we have discussed so far involve ignoring, or leaving out, certain considerations. As a result, the control rules that are generated are not guaranteed to be useful. We note that there are other types of simplifying abstractions where the "simplification" results from restricting the theory to some easily analyzed special cases(Etzioni 1993; Etzioni & Minton 1992). This is an important technique for simplifying the explanation process, but not particularly germane to our argument.

[3] In some simple analytic learning systems, the theory used to derive control knowledge is identical to the theory used by the problem solver, but in such systems, all that is possible is a form of generalized caching, a very simple type of control.

[4] Obviously in certain limited situations it is possible to derive control knowledge that is guaranteed to be useful, but this is relatively uninteresting.

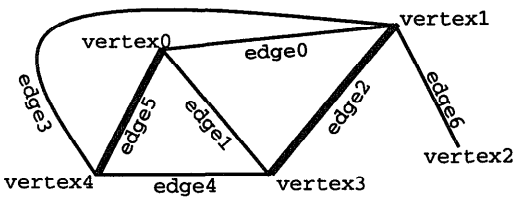

Figure 1: An instance of MMM with $K = 2$. A solution $E' = \{\text{edge2 edge5}\}$ is indicated in boldface.

Control Knowledge in Multi-TAC

In this section we describe some recent experiments with MULTI-TAC which further illustrate the points raised in the previous section. MULTI-TAC is a system that synthesizes heuristic constraint-satisfaction programs(Minton 1993; Minton & Underwood 1994; Minton 1996). The system has a small library of generic algorithm schemas, currently including a backtracking schema and an iterative-repair schema. Given one of these schemas, MULTI-TAC can specialize it to produce an application-specific program. As part of the specialization process, the system generates search control rules that can be incorporated into the schema. MULTI-TAC uses a set of sample problem instances – training instances – to evaluate the utility of the search control rules, in an effort to determine the best set of control rules. The system can be regarded as a learning system because it uses training instances to guide the specialization process. The process is not the same as in Prodigy, but is similar in some respects.

The experimental work that we describe here was done using MULTI-TAC's backtracking schema, which is based on the standard CSP backtracking algorithm. The backtracking algorithm operates by successively selecting a variable and then assigning it a value. Backtracking occurs when all values for a variable fail to satisfy the constraints. The backtracking schema includes several decision points, two of the most important being the variable and value selection points. MULTI-TAC synthesizes control rules for these decision points.

The results that describe here were produced as a part of an in depth study with a problem called Minimum Maximal Matching (MMM). MMM is an NP-complete problem described in (Garey & Johnson 1979). An instance of MMM consists of a graph $G = (V, E)$ and an integer K. The problem is to determine whether there is a subset $E' \subseteq E$ with $\mid E' \mid \leq K$ such that no two edges in E' share a common endpoint and every edge in $E - E'$ shares a common endpoint with some edge in E'. See Figure 1 for an example.

To formulate MMM as a CSP, we represent each edge in the graph by a boolean variable. If an edge is assigned the value 1, this indicates that it is in E', otherwise, it is assigned the value 0 indicating it is in $E - E'$. The constraints are specified as follows.

1. If $edge_i$ is assigned 1, then every $edge_j$ that shares an endpoint with $edge_i$ must be assigned 0.
2. If $edge_i$ is assigned 0, then there must exist an $edge_j$ such that $edge_i$ and $edge_j$ share an endpoint, and $edge_j$ is assigned 1.
3. The cardinality of the set of edges that are assigned 1 must be less than or equal to K.

In MULTI-TAC each of these three constraints is represented by a sentence in a form of first-order logic.

In the original experiments with this problem (Minton 1993), we created three different instance distributions for MMM and for each distribution, we compared programs synthesized by computer scientists for that distribution to programs synthesized by MULTI-TAC for the same distribution. MULTI-TAC's programs were generally competitive with the hand-coded programs, and in some cases, superior. However, one problem with the original study (pointed out by Ginsberg) is that the results might be deceiving. It is possible that if one picked a good pre-existing domain-independent algorithm and tried it on MMM, it might perform much better than *both* the handcoded programs and MULTI-TAC's programs. To address this, we recently compared MULTI-TAC's programs (synthesized in the original study) to some well-known generic algorithms: TABLEAU (Crawford & Auton 1993), GSAT (Selman, Levesque, & Mitchell 1992), and FC-D (forward checking + minimum domain ordering, a standard CSP algorithm). We found that MULTI-TAC outperformed these programs (Minton 1996), and we will briefly summarize some of the results of the comparison between MULTI-TAC and TABLEAU, the "second-place" finisher, simply to illustrate the utility of the control knowledge produced by MULTI-TAC. We will then consider why this knowledge would be difficult to derive solely through domain-independent logical analysis.

As in the original experiments, we tested the programs using 100 randomly-selected test instances from each distribution. For each distribution, we also used the same time bound (per instance) that was used in the original study, 10 seconds for the first two distributions and 45 seconds for the third.[5] Figure 1 shows a CPU-time comparison between MULTI-TAC's programs and an implementation of TABLEAU provided by Crawford (TABLEAU's developer). For each distribution, the first column shows the total time used on the 100 instances. Since the programs did not necessarily solve each instance within the time bound, the second column shows the number of solved instances.

[5] We did not include the time necessary for MULTI-TAC to synthesize its programs in the comparison, since as discussed in (Minton 1993), we assume that MULTI-TAC will only be used in applications where compile time is not a critical factor.

	Distribution1		Distribution2		Distribution3	
	CPU sec	num solved	CPU sec	num solved	CPU sec	num solved
Multi-TAC	5.9	100	43.5	99	515	92
Tableau	52.3	100	892.5	26	3002	43

Table 1: Comparison: MULTI-TAC vs. TABLEAU

The CPU-time results show that MULTI-TAC's programs outperformed TABLEAU, demonstrating the value of the control knowledge produced by MULTI-TAC. However, other than establishing this fact, the performance results are tangential to this discussion. The interesting part is why MULTI-TAC outperformed TABLEAU. (Out of necessity, we can only summarize the detailed results described in (Minton 1996).)

The first two distributions were relatively easy, as described in the original study (Minton 1993), and MULTI-TAC synthesized essentially the same program for both distributions. TABLEAU did not perform as well as MULTI-TAC on these distributions, but a closer analysis shows that TABLEAU was partly handicapped by the clause form representation that it employs. Representing the third MMM constraint in clause form involved the addition of a potentially large (but polynomial) number of *auxiliary variables* used to "count" the size of the subset E. The instances in the second distribution were particularly large, so there was a significant overhead for TABLEAU. However, if one looks at the search effort involved, it turns out that TABLEAU's heuristics were reasonably effective in the first two distributions. For instance, if TABLEAU is given 5 times more time on each instance in the second distribution, it can solve most of the instances.

In contrast, the third distribution was more difficult, even though the instances were smaller than those in the second distribution. Again, MULTI-TAC outperformed TABLEAU, but here we find that MULTI-TAC was actually searching much more effectively than TABLEAU; the overhead due to auxiliary variables was not sufficient to explain TABLEAU's difficulties on these instances. For instance, given 5 times more time, TABLEAU still performs relatively poorly.

Examining the program synthesized by MULTI-TAC for the third distribution, we find that the variable and value ordering heuristics MULTI-TAC used were very different than those it used for the first two distributions. For the first two distributions, MULTI-TAC used the following heuristics:[6].

value ordering : Try 1 before 0

variable ordering : Prefer edges with the most adjacent edges.

For the third distribution, MULTI-TAC used these heuristics:

value ordering : Try 0 before 1.

variable ordering :

1. Prefer edges that have no adjacent edges along one endpoint.
2. Break ties by preferring edges with the most endpoints such that all edges incident along that endpoint are assigned. (I.e, an edge is preferred if all the adjacent edges along one endpoint are assigned, or even better, if all adjacent edges along both endpoints are assigned).
3. If there are still ties, prefer an edge with the fewest adjacent edges.

Notice that the rules in the first set, considered individually, are contrary to the rules in the second set. (This was considered a surprising result when the original study was conducted.) Nevertheless, when each rule set is considered as a single entity, they make sense. Intuitively speaking, the programs for the first and second distribution operate by attempting to incrementally construct the subset of edges involved in the matching (the subset E), whereas the program for third distribution operates by attempting to construct the complement of this set ($E - E'$). We speculate that in the third distribution, because the graphs are more dense, the system could more accurately guess which edges should be in $E - E'$, which explains the change in strategy.

The differences in these rule sets illustrate the difficulty of deriving control information directly from domain knowledge. Here we have two sets of control rules for what normally would be considered the same domain (in standard AI terms); of course, in reality, the domains are not the same because the instances were selected from different distributions. Although the instance distribution is not normally considered domain knowledge, the experimental results illustrate that the distributions can be critical for determining what types of control rules are appropriate. (This can be even more dramatically demonstrated by running the program derived for the third distributions on the second distribution, or vice versa; In either case, the performance suffers, as shown in (Minton 1993).)

The Role of Simplifying Abstractions

G&G raise the possibility that probabilistic knowledge about the domain can be used to derive control rules. For instance, in one of their examples, they suggest that a problem solver could derive control knowledge

[6]When reading the heuristic rules, recall that each variable in MMM corresponds to an edge

by reasoning about how tightly constrained different subproblems are. Given our results, probabilistic information would indeed seem necessary, but the difficulty, as we have said earlier, is knowing what sort of probablistic domain knowledge will be *sufficient* for producing good control rules (and formalizing the requisite inference rules). In fact, MULTI-TAC does use concepts such as "most constrained first" to derive control rules; however, these concepts are only used as simplifying abstractions, and the resulting control rules, while plausible, are certainly not guaranteed to be effective.

For example, MULTI-TAC derives variable-ordering rules by attempting to operationalize generic heuristics such as "choose the most-constraining variable first" and "choose the most-constrained variable first".[7] Consider the variable-ordering rule "Prefer edges with the most adjacent edges first" which, as discussed above, was used for the first and second distributions. MULTI-TAC determines that this is a plausible rule by reasoning about the first MMM constraint (i.e., if an edge has value 1, then all adjacent edges must have value 0). The system reasons that, according to the first constraint, a variable with many neighbors is more likely to constrain the most other variables. A rule which makes exactly the opposite recommendation, "Prefer edges with the fewest adjacent edges first" (used for the third distribution) is produced by reasoning about the second constraint (i.e., if an edge is assigned 0 then some adjacent edge must be assigned 1). The system reasons that, according to the second constraint, an edge with few neighbors will have a more constraining effect on the other variables.

"Most Constraining", as defined in MULTI-TAC, is a simplifying abstraction in several respects. First, the constraints are analyzed individually, which simplifies the analysis greatly. Second, the notion of "constraining" that we use is very simplistic – variable A can constrain variable B if instantiating variable A can eliminate any of variable B's possible values. Third, the system relies on a completely qualitative style of reasoning, rather than on a quantitative method. (As illustrated in our example, the system can infer that "more neighbors → more constraining", but it cannot determine "how much" more constraining.) As a result tradeoffs cannot be logically analyzed.

It is because of such simplifications that the derived control rules must be regarded as plausible, and why empirical utility evaluation is necessary to determine which rules really are sensible. (Of course, even if the recommendations made by the control rules were guaranteed to be correct, utility evaluation would still be required to deal with the considerations outlined in the previous section, i.e., their match cost, application frequency, and benefit.)

[7] MULTI-TAC includes two different learning components for producing control rules (Minton 1996), but for pedagogical purposes we focus on only one of them here, the analytic mechanism. Our points could be illustrated with the other component as well.

The Role of Utility Evaluation

As we have discussed, empirical utility evaluation enables a system to bypass very complex analyses that would be impractical to expect a system to carry out – analyses that even experts might have difficulty with. A relatively simple example of the complexities we are referring to is illustrated by the rule sets above. If one thinks about the two rule sets, and why they each work, it should be apparent that the variable and value-ordering rules interact – the choice of value-ordering rule affects the choice of variable-ordering rule, and vice-versa. This is, in our opinion, a potentially complex issue, and we are not aware of any analysis, probabilistic or otherwise, of this phenomenon in the existing literature. But clearly this is the type of issue that must be completely analyzed before there can exist a fully capable domain-independent reasoning system for deriving control knowledge.

It is instructive to consider what sort of theory would be required to be able to *prove* that, in any given circumstance, a variable is most likely to be most constraining. Presumably one would need to prove that after instantiating that variable, the average size of the explored search tree would be less than that for any other variable. It seems unlikely that there exists any tractable way of generating such proofs.

In general, the complexities involved in proving, through logical inference, that a search control rule is both definitely correct and definitely useful are daunting. Thus is why we believe empirical evaluation has an important role to play in finding good control rules.

Conclusions

This paper has made three contributions. First, we have pointed out the importance of simplifying abstractions in deriving control rules (a point that has not received the attention it deserves). While much of the work with PRODIGY-EBL and MULTI-TAC depended heavily on simplifying abstractions to generate control rules, previous papers on these systems focused on other points. G&G do not discuss the role of simplifying abstractions when discussing how control rules could be derived. We contend that simplifying abstractions often play a critical role in systems that derive control knowledge for non-trivial domains.

Second, we have argued that, since the logical derivations used to produce control rules often involve simplifications, empirical utility evaluation can play an valuable role in determining whether a control rule is actually useful.

Finally, we have critically examined G&G's claim that domain-dependent control knowledge has no place in declarative systems. While their ideas are intriguing, their proof that domain-dependent control knowledge can be divided into a domain-dependent modal

fact and a domain-independent control rule establishes only a much, much narrower point that is only of technical interest. The more interesting practical question is, as they say, more suited for experimental studies. We have not disproven their claim – in fact, it's not clear that such a claim can be proven or disproven – but we have presented concrete, implemented examples challenging their assumptions. In particular, we argued that in many cases we will need to empirically evaluate the utility of control rules. In such cases, we cannot easily dispense with the rules themselves.

While this paper has focused on our differences with G&G's conclusions, we do not wish to draw attention away from their substantial contributions. G&G contend, rightly in our view, that domain-independent principles play a primary role in acquiring control knowledge. The questions we have investigated here, and which deserve further study, are concerned with the nature of those principles and how the reasoning process can and should be engineered in practical systems.

Acknowledgements

We are indebted to Matt Ginsberg, for many provocative (and enjoyable) conversations, and to Jimi Crawford, for providing the Tableau implementation we used our experiments.

References

Bhatnagar, N., and Mostow, J. 1994. On-line learning from search failures. *Machine Learning* 15(1).

Crawford, J., and Auton, L. 1993. Experimental results on the crossover point in satisfiability problems. In *AAAI-93 Proceedings.*

Eskey, M., and Zweben, M. 1990. Learning search control for constraint-based scheduling. In *AAAI-90 Proceedings.*

Etzioni, O., and Minton, S. 1992. Why ebl produces overly-specific knowledge: A critique of the prodigy approaches. In *Proceedings of the Ninth International Machine Learning Conference.*

Etzioni, O. 1993. Acquiring search control knowledge via static analysis. *Artificial Intelligence* 62(1).

Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co.

Ginsberg, M., and Geddis, D. 1991. Is there any need for domain-dependent control information? In *AAAI-91 Proceedings.*

Gratch, J. 1995. On efficient approaches to the utility problem in adaptive problem solving. Technical Report 1916, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Katukam, S., and Kambhampati, S. 1994. Learning explanation-based search control rules for partial order planning. In *AAAI-94 Proceedings.*

Minton, S., and Underwood, I. 1994. Small is beautiful: A brute-force approach to learning first-order formulas. In *AAAI-94 Proceedings.*

Minton, S.; Carbonell, J.; Knoblock, C.; Kuokka, D.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem solving perspective. *Artificial Intelligence* 40:63–118.

Minton, S. 1988. *Learning Search Control Knowledge: An Explanation-based Approach.* Boston, Massachusetts: Kluwer Academic Publishers. Also available as Carnegie-Mellon CS Tech. Report CMU-CS-88-133.

Minton, S. 1993. Integrating heuristics for constraint satisfaction problems: A case study. In *AAAI-93 Proceedings.*

Minton, S. 1996. Automatically configuring constraint satisfaction programs: A case study. *Constraints* 1(1).

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *AAAI-92 Proceedings.*

Smith, D. 1985. *Controlling Inference.* Ph.D. Dissertation, Computer Science Dept., Stanford University.