A Methodology for Modeling Sophisticated Problem Solvers

Article · July 1995

Source: CiteSeer

CITATIONS
0

READS
11

2 authors, including:

Victor Lesser
University of Massachusetts Amherst
519 PUBLICATIONS **17,942** CITATIONS

A Methodology for Modeling Sophisticated Problem Solvers *

Robert C. Whitehair and Victor R. Lesser
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
413-545-3444(voice)
413-545-1249(fax)
Email: whitehair@cs.umass.edu

January 5, 1995

Abstract

This paper presents the use of the *IDP formalism* for analyzing the relationship between the performance of search-based interpretation problem solving systems and the inherent properties, or *structure*, of problem domains in which they are applied. Specifically, we show how to formally represent and statistically evaluate the effectiveness of meta-level control strategies that are based on abstractions, approximations, and reformulations of a problem domain. Experimental results are presented verifying the predictions from the formal analysis techniques.

Keywords: mathematical foundations, problem solving architectures, control, search,

*This work was supported by the Office of Naval Research contract N00014-92-J-1450. The content does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred. This paper has not already been accepted by and is not currently under review for a journal or another conference. Nor will it be submitted for such during IJCAI's review preiod.

1 Introduction

This paper presents the *IDP formalism* for analyzing the relationship between the performance of search-based interpretation problem solving systems and the inherent properties, or *structure*, of problem domains in which they are applied. More specifically, the IDP formalism is intended to analyze the assumptions (implicit and explicit) that meta-operators make about the structure of a domain. The formalism consists of two components, one, IDP_G , for modeling the statistical properties of the domain phenomena that generate specific problem instances and another, IDP_I for modeling the structure and characteristics of an interpretation problem solving architecture. Though distinct, IDP_G and IDP_I are closely connected. By providing a formal model of a domain, IDP_G establishes the statistical characteristics of the phenomena a problem solver will encounter.

For example, consider a a problem solving architecture that includes a costly mechanism for filtering out certain kinds of noise. The IDP_G specification defines properties of the noise such as the frequency of its occurance, its spacial distribution, etc. This information is a necessary basis for evaluating the cost effectiveness of the control mechanism since the benefits derived from the use of the mechanism are a function of properties of the noise such as frequency, etc. If the overhead associated with the mechanism is very costly and the frequency of noise is low, the mechanism may *increase* the computational cost of problem solving or its performance may be inferior to a less effective technique that has a lower overhead. In general, this capability is critical for comparative experiments involving multiple problem solving architectures applied in the same domain or in multiple, alternative domains. In addition, IDP_G provides a basis for defining a problem solver's architecture, as we demonstrate in this paper. This work represents an initial step needed to formalize complex search processes, such as those associated with blackboard systems [1], that are used in sophisticated interpretation systems.

The IDP formalism models interpretation *domain theories*[1], in terms of four feature structures: *component* (or *syntax*), *utility* (or *credibility*)[2], *probability* (or *distribution*), and *cost*. The different feature structures are represented in terms of a *domain grammar* and functions associated with production rules of the grammar. Search paths are represented as *interpretation trees* of the grammar. By analyzing the statistical properties of the interpretation trees of a domain grammar, it is possible to determine characteristics of problem solving such as the expected cost for a random problem instance, the expected credibility of a solution, etc.

An important aspect of the IDP formalism is that it supports a unified representation of both *meta-operators*[1] and domain processing. This unified representation is based on viewing sophisticated control architectures as mechanisms that evaluate problem solving actions by selectively applying a process that examines a search

[1] A domain theory is the computational theory that is the basis for a problem solver's functionality.

[2] In [5], credibility structures are formally linked to the semantics associated with full and partial interpretations. Thus, a full or partial interpretation that has a high credibility can intuitively be thought of as having a highly consistent semantic interpretation and a full or partial interpretation that has a low credibility can intuitively be thought of as having an inconsistent or incomplete semantic interpretation.

path's relationship with other, possibly interacting, search paths [1]. (Relationships are based on the distribution of domain events and are statistical in nature. For example, a relationship might indicate that two search paths lead to the same final result 50% of the time.) Furthermore, the process of examining a search path's relationships are viewed as a *distinct search operation* and not as part of the control architecture. Interrelationships between search paths are exploited through abstract or approximate states that are explicitly created by search operators and not by a monolithic control process. In this way, the cost of applying meta-level control can be fully accounted for and the use of meta-level control can potentially be turned on and off dynamically based on the emerging state of the search space (i.e., this representation enables meta-meta-level control).

The primary focus of this paper will be on IDP$_I$ and the details of IDP$_G$ are left to [5]. The next section presents how the IDP formalism can be used to represent interpretation meta-operators in a unified representation with domain processing. Section 3 describes the derivation of a technique for calculating the expected cost of problem solving from the resulting grammar specification. Section 4 presents experimental results that confirm this calculation.

2 Interpretation Problems and the IDP Formalism

The IDP formalism describes a class of problems, *Interpretation Problems*, in terms of a search process where domain problem solving can be represented in terms of classes of possible intermediate states, the features associated with each individual state instantiation, the operators applicable to each state, and the cost of the operators. In more formal terms, as defined in [5], a problem solver is characterized by the four-tuple $< \mathcal{S}, \Omega, \omega, \Phi >$, where;

$\mathcal{S}$ = the *start state*; $\mathcal{S}$ is defined by the input data to the problem solver and the initial values of any relevant *characteristic variables, CVs*. CVs represent the properties of intermediate results and final solutions that define a particular domain.

Ω = the *base search space* with associated CVs and operators. Ω corresponds to the traditional notion of a search space. It is defined by CVs that specify the characteristics of individual states, operators that map one state to another, and functions of CVs that define final states. In terms of an IDP, Ω is defined by the interpretation grammar.

ω = a set of *projections*, or *abstractions*, of the base search space, each with their associated CVs and operators. Final states are those from the base space that can be reached via mapping operators. A given search space projection ω_i is defined by two sets of operators, $OP_{(\Omega,\omega_i)}$ and $OP_{(\omega_i,\omega_i)}$. $OP_{(\Omega,\omega_i)}$ is the set of operators that map states from Ω to states in ω_i. $OP_{(\omega_i,\omega_i)}$ is the set of operators that map states in ω_i to other states in ω_i. As with Ω, each state in a projected search space is characterized by a set of CVs.

Φ = a set of *mapping functions* from projection spaces back to the base search space. Functions in Φ can be thought of as the mechanisms that map constraints from

an abstract space, ω_i, back to the base search space, Ω. This can be done by creating new states in Ω, or by modifying existing states.

The definition of Ω is derived from the manner in which the IDP$_G$ formalism represents a problem domain in terms of a context-free grammar. Search states belong to classes corresponding to terminals and nonterminals of the grammar. Search operators correspond to the production rules of the grammar. During problem solving, there could be many instances of a given class, each associated with a different search state. Given the correspondance between elements of an IDP$_G$ grammar and states in Ω, it is possible to compute relationships between states by computationally determining relationships between elements of the grammar. As will be discussed in Section 3, these relationships can be used in analyzing the expected performance of a problem solving architecture. Formally,

Definition 2.1 *An IDP Grammar is a grammar,* $G_I =< V, N, SNT, S, P >$, where V is the set of terminal symbols, N is the set of nonterminal symbols, SNT is the set of *solution-nonterminal*[3] symbols that correspond to final states, S is the start symbol for the grammar, and P is the set of context-free production rules.

The IDP formalism describes a class of problems, *Interpretation Problems*, in terms of a search process where, given an input string X, a problem solver attempts to find the most *credible* (or "best") explanation for X from the set I of all possible interpretations. Interpretation is a form of *constructive problem solving* based on *abductive inferencing* [1]. Interpretation problems are similar to *classification* problems and to a more distant form of problem solving, *parsing*. In interpretation problems, the set I is constructed dynamically. This is in contrast to classification problems where I is preenumerated and the problem solving task involves only the identification of the best element of I.

I is a potentially enormous set. Each element of I can be thought of as an instantiation of an SNT or as an interpretation tree for a specific SNT. For many domains, I can be specified in a naturally structured way and it is this structure that is exploited by control architectures to reduce the number of elements of I that must be generated or to otherwise increase the efficiency with which I is specified. Of particular interest are situations where I is finite and can be defined as the language generated by a grammar, G, and where the evaluation function, f, used during problem solving is recursively defined for interpretation trees i in I. In these situations, interpretations take the form of derivation trees of X and the constructive search operators used in interpretation problems are viewed as production rules of G. G, therefore, defines how interpretations are decomposed. For example, the production rule $p \rightarrow n_1 n_2 n_3$ might correspond to the interpretation "p is composed of an n_1, an n_2, and an n_3." Given an n_1, an n_2, or an n_3, a problem solver may invoke the search operator o_p to try and generate a p. Each syntactic rule of the grammar is associated with a corresponding semantic process which determines the actual "meaning" of an interpretation. This representation is similar to that used in the *Composite Decision Process (CDP)* model of Kanal and Kumar [4] and the semantic grammars used by Fu [2].

[3] In the IDP formalism, a potential solution is referred to as an SNT. The uses and advantages of the SNT representation are discussed in [5].

grammar rule	distribution	credibility	cost
0.1 S → A	ψ(0.1) = 0.2	$f_{0.1}(f_A)$	$g_{0.1}(g_A)$
0.2 S → B	ψ(0.2) = 0.2	$f_{0.2}(f_B)$	$g_{0.2}(g_B)$
0.3 S → M	ψ(0.3) = 0.2	$f_{0.3}(f_M)$	$g_{0.3}(g_M)$
0.4 S → N	ψ(0.4) = 0.2	$f_{0.4}(f_N)$	$g_{0.4}(g_N)$
0.5 S → O	ψ(0.5) = 0.2	$f_{0.5}(f_O)$	$g_{0.5}(g_O)$
1. A → CD	ψ(1) = 1	$f_1(f_C f_D, \Gamma_1(C,D))$	$g_1(g_C g_D, C(\Gamma_1(C,D)))$
2. B → DEW	ψ(2) = 1	$f_2(f_D f_E f_W, \Gamma_2(D,E,W))$	$g_2(g_D g_E g_W, C(\Gamma_2(D,E,W)))$
3.0 C → fg	ψ(3.0) = 0.5	$f_{3.0}(f_f, f_g, \Gamma_{3.0}(f,g))$	$g_{3.0}(g_f, g_g, C(\Gamma_{3.0}(f,g)))$
3.1. C → fgq	ψ(3.1) = 0.5	$f_{3.1}(f_f f_g f_q, \Gamma_{3.1}(f,g,q))$	$g_{3.1}(g_f g_g g_q, C(\Gamma_{3.1}(f,g,q)))$
4. E → jk	ψ(4) = 1	$f_4(f_j f_k, \Gamma_4(j,k))$	$g_4(g_j g_k, C(\Gamma_4(j,k)))$
5.0 D → hi	ψ(5.0) = 0.5	$f_{5.0}(f_h f_i, \Gamma_{5.0}(h,i))$	$g_{5.0}(g_h g_i, C(\Gamma_{5.0}(h,i)))$
5.1. D → rhi	ψ(5.1) = 0.5	$f_{5.1}(f_r f_h f_i, \Gamma_{5.1}(r,h,i))$	$g_{5.1}(g_r g_h g_i, C(\Gamma_{5.1}(r,h,i)))$
6.0 W → xyz	ψ(6.0) = 0.5	$f_{6.0}(f_x f_y f_z, \Gamma_{6.0}(x,y,z))$	$g_{6.0}(g_x g_y g_z, C(\Gamma_{6.0}(x,y,z)))$
6.1. W → xy	ψ(6.1) = 0.5	$f_{6.1}(f_x f_y, \Gamma_{6.1}(x,y))$	$g_{6.1}(g_x g_y, C(\Gamma_{6.1}(x,y)))$
7. f → (s)	ψ(7) = 1	$f_7(f_{(s)}, \Gamma_7((s)))$	$g_7(g_{(s)}, C(\Gamma_7((s))))$

grammar rule	distribution	credibility	cost
8. j → (s)	ψ(8) = 1	$f_8(f_{(s)}, \Gamma_8((s)))$	$g_8(g_{(s)}, C(\Gamma_8((s))))$
9. g → (s)	ψ(9) = 1	$f_9(f_{(s)}, \Gamma_9((s)))$	$g_9(g_{(s)}, C(\Gamma_9((s))))$
10. k → (s)	ψ(10) = 1	$f_{10}(f_{(s)}, \Gamma_{10}((s)))$	$g_{10}(g_{(s)}, C(\Gamma_{10}((s))))$
11. h → (s)	ψ(11) = 1	$f_{11}(f_{(s)}, \Gamma_{11}((s)))$	$g_{11}(g_{(s)}, C(\Gamma_{11}((s))))$
12. x → (s)	ψ(12) = 1	$f_{12}(f_{(s)}, \Gamma_{12}((s)))$	$g_{12}(g_{(s)}, C(\Gamma_{12}((s))))$
13. i → (s)	ψ(13) = 1	$f_{13}(f_{(s)}, \Gamma_{13}((s)))$	$g_{13}(g_{(s)}, C(\Gamma_{13}((s))))$
14. y → (s)	ψ(14) = 1	$f_{14}(f_{(s)}, \Gamma_{14}((s)))$	$g_{14}(g_{(s)}, C(\Gamma_{14}((s))))$
15. z → (s)	ψ(15) = 1	$f_{15}(f_{(s)}, \Gamma_{15}((s)))$	$g_{15}(g_{(s)}, C(\Gamma_{15}((s))))$
16. M → Y	ψ(16) = 1	$f_{16}(f_Y)$	$g_{16}(g_Y)$
17.0 Y → qr	ψ(17.0) = 0.5	$f_{17.0}(f_q f_r, \Gamma_{17.0}(q,r))$	$g_{17.0}(g_q g_r, C(\Gamma_{17.0}(q,r)))$
17.1 Y → qhri	ψ(17.1) = 0.5	$f_{17.1}(f_q f_h f_r f_i, \Gamma_{17.1}(q,h,r,i))$	$g_{17.1}(g_q g_h g_r g_i, C(\Gamma_{17.1}(q,h,r,i)))$
18. N → Z	ψ(18) = 1	$f_{18}(f_Z)$	$g_{18}(g_Z)$
19. Z → xy	ψ(19) = 1	$f_{19}(f_x f_y, \Gamma_{19}(x,y))$	$g_{19}(g_x g_y, C(\Gamma_{19}(x,y)))$
20. O → X	ψ(20) = 1	$f_{20}(f_X)$	$g_{20}(g_X)$
21.0. X → fgh	ψ(21.0) = 0.5	$f_{21.0}(f_f f_g f_h, \Gamma_{21.0}(f,g,h))$	$g_{21.0}(g_f g_g g_h, C(\Gamma_{21.0}(f,g,h)))$
21.1. X → fg	ψ(21.1) = 0.5	$f_{21.1}(f_f f_g, \Gamma_{21.1}(f,g))$	$g_{21.1}(g_f g_g, C(\Gamma_{21.1}(f,g)))$

(s) = signal data Γ_n(i,j,...) = semantic evaluation function for rule n $C(\Gamma_n(i,j,...))$ = cost of executing Γ_n(i,j,...)

Figure 1: Interpretation Grammar with Fully Specified Distribution, Credibility, and Cost Functions

This approach is significant because it can be used to define the component, credibility, distribution, and cost structures of a domain. The component structure is modeled directly by the rules of G. Thus, the component subproblems of A are defined by the right-hand-side (RHS) of a rule of the grammar, $p : A \rightarrow$(component subproblems). The full structure is specified recursively and subproblems specified by the nonterminals and terminals of a grammar correspond to states in the associated search space.

The credibility structure is defined recursively in terms of credibility functions associated with G's production rules. Credibility functions include consideration of the credibilities of the component elements and semantics. Given p. A → CD, the credibility of "A," f_{A}, is a function of the credibilities of "C" and "D" and the *semantic function*, Γ, that measures the degree to which "C" and "D" are semantically consistent. Semantics associated with each production rule determine the actual domain interpretation. The semantic functions will typically make use of grammar element attributes that are not used by syntactic functions. These attributes are represented using the *feature list convention* described by Gazdar, et al.[3]. For example, a semantic process in an acoustic vehicle tracking system might combine two partial vehicle tracks into a longer track using a very complex process that verifies the consistency of the frequency characteristics of the partial tracks and the properties of the resulting track such as acceleration and velocity constraints. Credibility functions are represented $f_{\mathrm{A}}(f_{\mathrm{C}}, f_{\mathrm{D}}, \Gamma_p(\mathrm{C},\mathrm{D}))$, where the subscript of Γ, p, is the number of the corresponding production rule from the grammar. In a natural language interpretation system, a semantic function might determine the degree to which the combination of a noun phrase and a verb phrase is meaningful.

Similarly, cost structures are recursive functions of component elements and the cost of the semantic function. For example, given p. A → CD, the cost of "A," g_{A} is a function of the cost of "C" and "D" and the cost of the semantic function $\Gamma_p(\mathrm{C},\mathrm{D})$. This is represented $g_{\mathrm{A}}(g_{\mathrm{C}}, g_{\mathrm{D}}, C(\Gamma_p(\mathrm{C},\mathrm{D})))$, where $C(\Gamma_p(i,j,...))$ is the cost of applying the semantic function Γ_P.

The probability structure is defined in terms of the function, ψ. For each $p \in P$, $\psi(p)$ represents the distribution of RHSs associated with p. This can be used

both to generate problem instances, in which case ψ models the likelihood of certain events happening in the environment, and to specify the actions available to a problem solver, in which case ψ models the problem solver's perception of and assumptions about the domain. For production p which decomposes to RHS_1, $\text{RHS}_2 \ldots \text{RHS}_m$, $\psi(p)$ models the frequency distribution with which these rules are used to generate problem instances. The frequency with which the rules are used in interpretation is a function of this distribution. Note that $\sum_n \psi(p.n) = 1$. ψ supports models of real-world phenomena such as uncertainty caused by noise, missing data, distortion and masking. Figure 1 illustrates an example interpretation grammar. The SNTs of this grammar are A, B, M, N, and O. Rules 3.1 and 5.1 are examples of noise which is represented by the additional elements q and r, respectively. Rules 6.1 and 21.1 are examples of missing data which is represented by the lack of z and h, respectively.

As illustrated in Fig. 1, alternative RHSs are distinct problem solving operators and this grammar defines Ω, the base search space. In the following subsections, we demonstrate how meta-level operators (meta-operators) can be represented in the IDP_I formalism.

2.1 Defining Bounding Functions

A *bounding function* is a type of meta-operator that will be used to demonstrate the analysis that can be performed with the IDP formalism. Given a partial interpretation, i, a bounding function determines the likelihood of i being included in a solution with a rating above some *threshold*. If it is unlikely that the partial interpretation leads to a solution rated above the threshold, all derivation paths including the partial interpretation are pruned. For example, given threshold t, for any partial interpretation, i, if $upperbound(i) < t$, then all paths from i can be pruned. $upperbound(i)$ can be defined as the maximum expected utility for any of the final states that can be reached along paths from i.

The IDP formalism represents meta-operators by defining new projections, ω, and associated mapping functions, Φ. Given a state, n, for which a bounding function is defined, a corresponding meta-state, n^b, is created in a bounding function abstraction space, ω_b. In addition, the production rule $p : n^b \rightarrow n$ is added to OP_{Ω,ω_b}. If the output of the bounding function indicates that extending interpretations using state n is pointless, the operator prunes the path by failing to generate the state n^b in ω_b and no further derivations using n are built. In the IDP formalism, a state is pruned by the semantic function, $\Gamma_{f_{n^b}}$, which returns a value of 0 for that state. If the output of the bounding function is within a range indicating that further interpretations using n should be constructed, the operator generates n^b in ω_b and processing continues. Finally, mapping functions are added to $OP_{\omega_b,\Omega}$ that correspond to the original grammar rules that included n but that have n replaced with n^b. The set OP_{ω_b,ω_b} is empty.

Figure 2 shows the results of adding bounding functions for states q and h to grammar G' from Fig. 1. The inclusion of a bounding function is represented by the addition of the rule $q^b \rightarrow q$. If, when applied, the bounding function operator determines that there is no point in extending state q, it fails to generate state q^b. This prevents the generation of state Y and makes the application of the operator

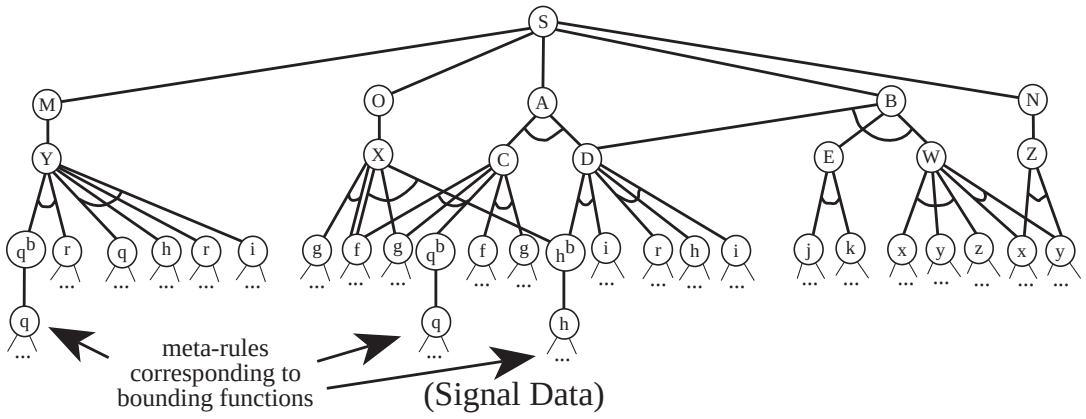

Figure 2: Example of Bounding Function Incorporated in Grammar G'

corresponding to the rule $C \rightarrow fgq^b$ unnecessary. This bounding function operator might be useful in a situation where state q is generated as the result of noise and has a credibility rating so low that any interpretation incorporating it would have an extremely low rating.

2.2 Representing Preconditions

Another example of a meta-operator is the concept of *preconditions*. Preconditions help focus search processes more effectively in sophisticated domains by performing some kind of preprocessing to more accurately judge the relative merits of executing an expensive problem solving action. For example, in a blackboard problem solver, a precondition might be executed before an operator is instantiated to determine if the data necessary to extend a partial result is present. This meta-level operation can be very effective in constructive search processes because each intermediate search state contains only local information about that particular state. Thus, if the rating of a potential search operation is heavily dependent on non-local information, the problem solver will not be able to accurately evaluate that action. The effectiveness of preconditions has been demonstrated in many interpretation systems including Hearsay II and the DVMT [1].

To clarify the use of preconditions, consider the production rule: *A → B C D*. If the problem solver has generated a search state "B," the operator associated with the production rule would be eligible for execution whether or not the required states C and D had been generated. In situations where C and D had not been generated, the operator would fail to generate an A. In contrast, using preconditions, the operator would not be eligible for execution until all the required syntactic elements were present.

To represent preconditions using the IDP formalism, we will define a projection space of satisfied precondition states. This space is created by operators of the form: $A^{op}_{[B,C,D]} \rightarrow$ B C D. Thus, a symbol with a superscript op will represent a state created by a successful precondition operator. op will correspond to the operator that will map the abstract state back to the base space. For clarity, the subscripted brackets will include the required syntactic elements used to create the precondition state. A production rule of this form will be created for each precondition operator.

1.	$S_{[f]}$	$\rightarrow$ $Tracks_{[f]}$	p=1
2.	$Tracks_{[f]}$	$\rightarrow$ $Tracks_{[f]}$ $Track_{[f]}$	p=0.1
		$\rightarrow$ $Track_{[f]}$	p=0.9
3.	$Track_{[f]}$	$\rightarrow$ I-$Track1_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ I-$Track2_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ P-$Track1_{[f,t,x,y]}$	p=0.10
		$\rightarrow$ P-$Track2_{[f,t,x,y]}$	p=0.10
		$\rightarrow$ G-$Track1_{[f,t,x,y]}$	p=0.15
		$\rightarrow$ G-$Track2_{[f,t,x,y]}$	p=0.15
4.	I-$Track1_{[f,t,x,y]}$	$\rightarrow$ I-$Track1_{[f,t+1,x+V+A,y+V+A]}$ $T1_{[f,t,x,y]}$	p=1
5.	I-$Track2_{[f,t,x,y]}$	$\rightarrow$ I-$Track2_{[f,t+1,x+V+A,y+V+A]}$ $T2_{[f,t,x,y]}$	p=1
6.	P-$Track1_{[f,t,x,y]}$	$\rightarrow$ P-$Track1_{[f,t+1,x+V+A,y+V+A]}$ P-$T1_{[f,t,x,y]}$	p=1
7.	P-$Track2_{[f,t,x,y]}$	$\rightarrow$ P-$Track2_{[f,t+1,x+V+A,y+V+A]}$ P-$T2_{[f,t,x,y]}$	p=1
8.	G-$Track1_{[f,t,x,y]}$	$\rightarrow$ G-$Track1_{[f,t+1,x+V+A,y+V+A]}$ G-$T1_{[f,t,x,y]}$	p=1
9.	G-$Track2_{[f,t,x,y]}$	$\rightarrow$ G-$Track2_{[f,t+1,x+V+A,y+V+A]}$ G-$T2_{[f,t,x,y]}$	p=1

Figure 3: Grammar Rules for Generating Patterns and Tracks

More formally, we say that the set of projection operators from the base space to the precondition abstraction space, $OP_{(\Omega,\omega_p)}$, is made up of precondition operators.

In addition, a mapping operator from the precondition projection space back to the base space will have the form: A $\rightarrow$A$^{op}_{[B,C,D]}$. This operator will correspond to the original operator except that it can only be applied to the states in the precondition space. The original operator will be replaced by these two rules. Formally, the set of mapping operators from the precondition abstraction space back to the base space is represented as $OP_{(\omega_p,\Omega)} \in \Phi$.

For the moment, we will not define any operators that create new states in the precondition space from states already in the space. Formally, this means that the set $OP_{(\omega_p,\omega_p)}$ is empty.

2.3 Representing Goal Processing

Another form of meta-operator that can be represented in the IDP formalism is the mechanism of goal processing developed in the DVMT blackboard problem solver and testbed [1]. Goal processing is used to increase the rating of problem solving activities that are related to extending highly-rated partial results. For example, the DVMT domain is a vehicle tracking problem based on acoustic sensor data. The problem solver's input consists of preprocessed sensor data gathered from a single contiguous region. The problem solver then processes the data in an attempt to identify the type of vehicle that generated the signals and the path, or track, it traversed through the region. Given a partial track with a high-rating, the focus of control goal processing would increase the ratings of operators that would generate the low-level data needed to extend the track. This would cause the low-level data to be generated sooner than it otherwise would have been, allowing the problem solver to extend the highly-rated partial track hypothesis.

To understand the benefits associated with this form of goal processing, consider a problem solver that combines it with dynamic pruning operators. Assuming that the pruning operators eliminate paths from consideration by comparing them with existing full interpretations, the more quickly the problem solver can construct a highly-rated full interpretation, the less total work is likely to be required. Given a partial track with a high-rating, it may be desirable to extend the track as quickly as possible to a full solution to maximize its pruning potential. However, if the data needed to

G.5.1. Goal-V$_{[t+1]}$ $\rightarrow$ I-Track1$_{[t]}$
G.5.2. Goal-V$_{[t-1]}$ $\rightarrow$ I-Track1$_{[t]}$

Figure 4: Meta-Level Operators for Focus-of-Control Goal Processing

M.G.5. $\emptyset$ $\rightarrow$ Goal-V$_{[t]}$
M.G.6. $\emptyset$ $\rightarrow$ Goal-G$_{[t]}$

Figure 5: Mapping Operators for the Goal Projection Space

extend the partial track is not available, the partial track cannot be extended. Focus-of-control goal processing is used to increase the rating of intermediate operators that will generate data needed to extend the partial track.

From a more general perspective, focus-of-control goal processing is a mechanism that allows a problem solver to selectively rerate problem solving operators based on their relationships with other, dynamically changing operators. As discussed in [5], one of the characteristics of sophisticated domains is that partial search paths are often related and the structure of these relationships can sometimes be exploited to increase the efficiency of problem solving. Exhaustive processing of these relationships during each problem solving cycle is infeasible because the number of relatiohships is combinatoric in the number of partial search paths and each relationship could involve arbitrarily complex processing. Focus-of-control goal processing restricts the search for and processing of relationships to cover only those relationships for which it is cost effective. This is a very general technique that is likely to have applicability in a wide range of domains.

Figure 3 shows production rules from a DVMT IDP$_G$ grammar. These rules are used to generate vehicle tracks and patterns of vehicle tracks. These rules also correspond to operators in Ω for the associated problem solving system [5]. Example goal processing operators are shown in Fig. 4. Operator G.5.1 is used to raise the ratings of operators that will generate data needed to extend the partial track forward in time. Similarly, G.5.1 is used to extend the partial track backwards in time.

Like the precondition operators, the goal processors define a projection space. Example mapping operators for this space are shown in Fig. 5. The mapping operators are applied to the states of the goal projection space. The $\emptyset$ notation on the LHS of the production rules indicates that no new states are generated in the base space. Instead, the ratings for existing states and instantiated operators is modified to reflect the information contained in the goal state. As with the precondition abstraction space, there are no operators in the set $OP_{(\omega_{goal},\omega_{goal})}$. Thus, in this example, there are no operators that create new goal states from existing goal states.

3 Quantitative Analysis with IDP Models

By unifying the representation of domain problem solving actions and meta-operators, the IDP formalism can be used as an analytical tool for prediction and explanation. This is demonstrated here with a derivation for the calculation of $E(C)$, the expected cost of a single problem solving instance for a given domain. E(C) is an important analysis tool because it provides a measure for comparing the performance of different

problem solving architectures applied to the same domain. E(C) can also be used as a measure of the performance of a specific problem solving architecture applied to a variety of different domains. Though generally useful, E(C) may not be the most appropriate measure in all circumstances. However, similar measures can be defined for minimum or maximum problem solving cost, etc.

E(C) is calculated for a problem solver that terminates when all states with non-zero credibility have been pruned or fully expanded. This derivation is verified in Section 4. The derivation of $E(C)$ will be valid for for two different scenarios. In one, the grammar specifying the occurrence of domain events, IDP_G, and the grammar defining the operators available to a problem solver and the problem solver's perception of the domain, IDP_I, are identical. In the other, IDP_G and IDP_I are different.

The calculation of E(C) is based on the expected cost associated with the search states that are generated. These states correspond to elements of the grammar and it is important to realize that *each element of the grammar may have multiple search state instantiations.* For example, in a given problem instance, element "A" of the grammar may correspond to many distinct search states. If a specific element of the grammar has an *expected frequency* of five, it means that, on average, five instantiations of that element of the grammar are made during an interpretation, each instantiation being a unique state in the corresponding search space. These instantiations (or states) are distinguished by the individual characteristics of the search paths that lead to the creation of the state. Although redundant paths lead to the same state instantiation representing the same interpretation or partial interpretation, other search paths, though they correspond to the same element of the IDP grammar, represent distinct interpretations or partial interpretations.

In Fig. 6, signal data leads to the generation of states representing multiple instantiations of the same IDP element. The signal data leads to the generation of the six interpretations shown. There are multiple instantiations of the partial result C that are shown in Figs. 6.a and 6.b as C^1 and in Figs. 6.b and Figs. 6.d as C^2. The differences between C^1 and C^2 are seen clearly in the figure as differences in the subtrees that were used to generate the specific instantiations of C. (This would be implemented as differences in the characteristic variable "supporting data" associated with each of the instantiations of C.) Similarly, there are two distinct instantiations of the partial results Y and D. The different partial result instantiations are used to generate four distinct instantiations of A and two distinct instantiations of the M. Finally, there are a total of six different interpretations of the data. This is shown as six different instantiations of S.

The following concepts are needed to define the calculation of $E(C)$.

Definition 3.1 *Sample Set, S_{sample}:* A set of specific problem instances generated using IDP_G.

Definition 3.2 *Sample Set Weightings, w_i:* A set of weighting factors for the elements of the Sample Set. The weighting of an element represents its likelihood is equal to the product of the ψ values of the grammar rules used to generate the element.

Definition 3.3 *Base Frequency for n, $F^B_{n,i}$:* The expected frequency of the creation of states corresponding to element $n \in \text{IDP}_I$ for a given sample $i \in S_{sample}$, is

interpretation trees derived from signal data “fqg rhi”:

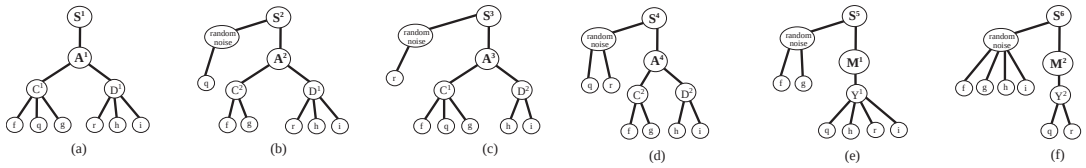

Figure 6: Signal Data Leading to Multiple IDP State Instantiations

determined from the expected frequencies of n's children, where a "child" of element n is a terminal or nonterminal element that appears on the RHS of one of n's production rules. Given, p, for which n is the left-hand-side element, $F^B_{n,i}$ is calculated in two steps; 1) The expected frequency of n from each of the RHSs of p is calculated as the product of the expected frequencies of the elements of the RHSs, or $\prod_j F_{e_j}$, where each $e_j \in$ RHS. 2) The expected frequencies from each of the RHSs of p are summed. In the case where n is a terminal symbol, its frequency corresponds to the number of occurrences in sample i. Combining functions for various meta-operators have also been defined [5] .

As shown in Fig. 2, for nonterminal C, there is one production rule with two RHSs; C $\rightarrow$ fqg | fg. The frequencies of f, g, and q are all 1 and the base frequency of C associated with each RHS is $(1 * 1 * 1)$ and $(1 * 1)$ respectively. The base frequency of C derived from the rule is the sum of these frequencies, or $(1 + 1)$. Two instantiations of C corresponding to these rules will be generated. Similarly, for nonterminal element A, there is a single rule with one RHS; A $\rightarrow$ CD. This results in a base frequency for A of $(2 * 2)$, since the frequency of D is two. This will result in four instantiations of A.

When pruning operations are available, the base frequency calculation must factor in the probability that a state is pruned based on its credibility. If other pruning operations are available, such as an operation that would prune a state based on its probability of being included in a final solution, they would be factored into the calculation of frequency in a similar way.

Definition 3.4 *Expected credibility of* n, $\mu_{Cred}(n) = \sum_i \mu_{f_{p.i}} * \psi(p.i)$, where $\mu_{f_{p.i}}$ is the expected credibility of the evaluation function $f_{p.i}$ and $\psi(p.i)$ is the distribution of the rules $p.i \in$ IDP$_I$. The experiments in this paper assume a distribution for the credibility functions $\Gamma_{p.i}$ that is normal with mean equal to the average of the means of the inputs. This enables the recursive calculation of the expected credibility of each element of the grammar based on the expected credibilities of an element's children.

Definition 3.5 n*'s Expected standard deviation from* $\mu_{Cred}(n)$, $\sigma_{Cred}(n) = ftn(\sigma_{f_{p.i}})$, where ftn is a function of the variances of the credibility rules, $f_{p.i}$ and is based on the standard equation $Var(X + Y) = VarX + VarY + 2 * Cov(X, Y)$. In this paper, variances for the credibility functions Γ_p are assumed to be equal to sum of the variances of the inputs. (The standard deviation is then the square root of the variance.) The credibilities and variances of siblings are assumed to be independent, and this simplifies to $Var(X + Y) = VarX + VarY$. As with expected credibility, $\sigma_{Cred}(n)$ can be calculated recursively for each element of the grammar.

Definition 3.6 *Pruning Modifier for* n, $Pf_n = (1 - P(Credibility(n) \leq T))$, where $Credibility(n)$ is the credibility of the search state corresponding to n that is determined dynamically at run time, and T is the pruning threshold[4]. Given normally distributed credibilities, this value can be calculated from $\phi(\frac{T-\mu_{Cred}(n)}{\sigma_{Cred}(n)})$, where ϕ is available from standard probability theory and is equal to $\frac{1}{\sqrt{2\pi}} * e^{-x^2/2}$.

Definition 3.7 *Expected frequency for element* n, $F_n = \sum_i w_i * (F_n^B * (1 - Pf_n))$, where $i \in S_{sample}$. F_n is calculated by modifying F_n^B to reflect pruning actions, multiplying by the sample weight to normalize the value, then summing all values.

It is now possible to calculate $E(C)$ by multiplying the expected cost associated with each search state by the frequency of the state and summing for all states.

Definition 3.8 *Expected cost associated with search state* n, $E_{cost}(n) = \sum_p \bar{g}_p$, where p is a production rule, $n \in$ some RHS of p, and $\bar{g}_p$ is the expected cost of applying the search operator corresponding to p. To determine the expected cost associated with a state, the expected costs of all the operators that can be applied to the state are summed. An operator can be applied to a state if the grammar element associated with the state appears in any of the RHSs of the grammar production rule associated with the operator.

Definition 3.9 *Expected cost of problem solving*, $E(C) = \sum_n F_n * E_{cost}(n)$. The expected cost of problem solving is determined by summing, for each element of the grammar, the expected frequencies of the grammar elements multiplied by the expected cost associated with the corresponding search states.

4 Experimental Verification

An experimental testbed consisting of a problem instance generator and a problem solver has been constructed to verify the IDP formalism. The testbed uses the specification of the problem domain, IDP_G, to generate problem instances. The problem solver uses its model of the problem domain's structure, IDP_I, to interpret each problem instance. Each production rule of the grammar has a credibility function associated with it that is used to generate ratings of intermediate and final problem solving states. The objective strategy that constitutes the basic control component of the problem solver is a simple *best-first* algorithm that attempts to generate a least cost interpretation based on a perspective of problem solving that relies solely on *local* information. i.e., it does not take into consideration the effects that an operator may have on other operators that are applied during subsequent processing. Credibility is calculated dynamically and the domain characteristics that change from run to run are represented with the feature list convention [3].

Experimental results are shown in Table 1. For each experiment, the column "Grammar" indicates the domain and problem solver specification used. $E(C)$ indicates the expected cost of problem solving in the domain based on an analysis of the grammar. "Avg. C" shows the actual average cost of problem solving. The "Sig"

[4][5] discusses bounding functions with thresholds that are determined dynamically. In these situations pruning modifiers are based on T equal to the expected credibility of a "correct" result for a given problem instance.

column indicates whether any difference in the Expected Cost and the Average Cost is statistically significant[5] The last column shows the number of correct answers that were found.

The experimental results confirm the accuracy of the analytic derivation of $E(C)$ from Section 3. Most importantly, the results demonstrate that the analysis is correct *even for situations where the problem solver's view of a domain differs significantly from the actual structure of a domain.* This is significant because it demonstrates that the IDP formalism can be used during the design of a problem solving system. By approximating the structure of a domain where the problem solver is to be used, it may be possible to make preliminary judgments about the worth of different meta-operators and sophisticated control techniques. In addition, the IDP formalism can be used to explain why a particular control technique does or does not work in a particular domain. It can also be used to compare and contrast the performance of different control techniques in the same domain. Some of these analyses are demonstrated in the following discussion.

Experiments 1 – 8 demonstrate the effects on problem solving performance associated with altering the structure the domain. Experiment 1 is the base line to which other experiments can be compared. In experiments 2 and 3, the cost of problem solving goes down due to some search paths being pruned by the bounding functions. Note that the bounding functions also pruned some correct paths. This is shown in the last column where the percentage of correct paths is less than 100. In Experiment 4, the cost of problem solving goes up dramatically when the generation of problem instances is skewed to more credible events. This causes fewer paths to be pruned, and magnifies the costs associated with the use of bounding functions. Experiment 6 also demonstrates how a meta-operator can be used inappropriately. In this experiment, the actual credibility of problem instances is so low that many correct solutions are pruned.

Experiments 9 - 17 can be compared with Experiment 8 to observe the effects caused by altering the distribution and credibility functions in IDP_I so that they are different from those in IDP_I. In effect, this changes the problem solver's expectations. Experiments 12 through 16 are more representative of real world domains where a problem solver's model of a domain's structure is slightly off both in terms of modeling the distribution of noise and missing data and in terms of modeling the distribution of credibility. In real domains, it is probably the case that some rules overestimate the correct credibility ratings and some underestimate. The same can be said for a problem solver's model of distribution of missing data and noise. Experiment 17 is the most realistic because the expected inaccuracies in a problem solver's model of a domain both increase the cost of problem solving and decrease the number of correct answers produced.

[5]The calculation of significance does not include any consideration of the percentage of correct answers found. Determination of significance is based on a t test with a 95% confidence level. In each experiment, the hypotheses tested were $H_0 : \mu_{cost} = E(C)$ and $H_1 : \mu_{cost} \neq E(C)$ where μ_{cost} is the normalized population mean (each element of the population consisting of 50 problem solving instances), and $E(C)$ is the analytically predicted normalized population mean. For experiments labelled significant, the null hypothesis, H_0, is rejected.

Exp	Generation			Interpretation			$E(C)$	Avg. C	Sig	% C
	G	Dist	U	G	Dist	U				
1	1	even	0.5	1	even	0.5	201	203	N	100
2	1	even	0.5	2	even	0.5	189	187	N	80
3	1	even	0.5	3	even	0.5	180	181	N	80
4	1	skew	0.5	2	even	0.5	368	369	N	100
5	1	skew	0.5	3	even	0.5	198	198	N	96
6	1	even	0.25	3	even	0.5	157	156	N	50
7	1	even	0.75	3	even	0.5	194	193	N	92
8	1	even	0.5	3	even	0.5	191	190	N	96
9	1	skew1	0.5	3	even	0.5	195	195	N	91
10	1	skew2	0.5	3	even	0.5	189	190	N	98
11	1	even	0.5	3	even	low	179	179	N	94
12	1	even	0.5	3	even	high	203	202	N	98
13	1	skew1	0.5	3	even	low	191	191	N	88
14	1	skew1	0.5	3	even	high	209	210	N	98
15	1	skew2	0.5	3	even	low	185	185	N	95
16	1	skew2	0.5	3	even	high	198	199	N	98
17	1	skew1	0.5	3	even	bad	203	202	N	92

Abbreviations

Exp: Experiment
Generation: Description of IDP Domain Grammar used to generate problem instances.
Interpretation: Description of IDP Interpretation Grammar specifying the problem solver.
G: The problem solving grammar used;
1: G';
2: G' and bounding functions with cost 10,
3: G' and bounding functions with cost 1,
Dist: Distribution of Domain Events;
even: domain events evenly distributed
skew: distribution skewed to more credible events
skew1: distribution skewed to more noise and missing data
skew2: distribution skewed to less noise and missing data
U**;** expected problem instance credibility; 0.25, 0.5, 0.75: expected credibility;
low: problem solver rates "bad data" lower
high: problem solver rates "bad data" higher
bad: problem solver rates some "bad data" higher, and some "good data" lower
$E(C)$: Expected Cost of problem solving for given grammar
Avg. C: actual average cost for 100 samples of 50 random problem instances each
Sig: Whether or not the difference between expected cost and the
actual average cost was statistically significant, Y:yes; N:no.
% C: percentage of correct answers found

Table 1: Results of Verification Experiments – Set 1

5 Conclusion

The IDP formalism represents a necessary step toward developing a design theory for constructing sophisticated knowledge-based problem solvers. This paper demonstrates how it can be used to describe and analyze complex interpretation systems that use sophisticated control techniques. Analysis can both predict the performance levels of the problem solver using analytically derived closed form equations and explain the performance levels relative to the given domain structures.

This paper demonstrates how the representation of domain processing and meta-operators can be unified. This paper also presents experimental data validating one form of analysis derived for the unified representation. The experiments explore issues associated with the use of incorrect domain theories by examining the performance implications in situations where a problem solver has a model of a domain that differs from the actual domain. The experiments demonstrate that the effects of deviations in credibility structures and in distribution structures can predicted, explained, and understood in terms of IDP structures.

This work has provided insights into the nature of search based problem solving. In particular, domain problem solving and meta-operators can now be viewed from a unified perspective. This unified perspective has important implications for analyzing control architectures such as approximate processing, goal processing, and other meta-level control architectures. Using the IDP formalism, or other formal approaches, it is possible to compare and contrast alternative architectures in a range of different domains in order to develop an understanding of how a particular architecture might be generalized.

References

[1] Norman Carver and Victor Lesser. The Evolution of Blackboard Control. *Expert Systems with Applications*, 7(1), 1991. Special issue on The Blackboard Paradigm and Its Applications.

[2] King Sun Fu. *Syntactic Pattern Recognition and Applications*. PH, 1982.

[3] Gerald Gazdar, Geoffrey K. Pullum, and Ivan A. Sag. Auxiliaries and related phenomena in a restrictive theory of grammar. *Language*, 58(3):591–638, 1982.

[4] Vipin Kumar and Laveen N. Kanal. The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, Symbolic computation, chapter 1, pages 1–27. Springer-Verlag, 1988.

[5] Robert C. Whitehair and Victor R. Lesser. A Framework for the Analysis of Sophisticated Control in Interpretation Systems. Technical Report 93–53, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, 1993.