

VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking

Sandra Marcus, Jeffrey Stout, John McDermott

VT (vertical transportation) is an expert system for handling the design of elevator systems that is currently in use at Westinghouse Elevator Company. Although VT tries to postpone each decision in creating a design until all information that constrains the decision is known, for many decisions this postponement is not possible. In these cases, VT uses the strategy of constructing a plausible approximation and successively refining it. VT uses domain-specific knowledge to guide its backtracking search for successful refinements. The VT architecture provides the basis for a knowledge representation that is used by SALT, an automated knowledge-acquisition tool. SALT was used to build VT and provides an analysis of VT's knowledge base to assess its potential for convergence on a solution.

Due to software problems at the typesetter, the publication of this article in volume 8, number 4 was flawed. A corrected copy is reprinted here. —Ed.

In some cases, plausible guessing combined with the ability to backtrack to undo a bad guess can be the most efficient way to solve a problem (Stefik et al. 1983). Even least commitment systems such as MOLGEN (Stefik 1981a, 1981b) are sometimes forced to guess. In the course of designing genetics experiments, MOLGEN tries to avoid making a decision until all constraints that might affect the decision are known. In some cases, this postponement is not possible, and the system becomes stuck; none of the pending decisions can be made with complete confidence. In such a case, a decision based on partial information is needed, and such a decision might be wrong. In this case, a problem solver needs the ability either to backtrack to correct bad decisions or to maintain parallel solutions corresponding to the alternatives at the stuck decision point. However, if alternative guesses exist at each point, and there are many such decision points on each solution path, a commitment to examine every possible combination of alternatives proves unwieldy. Such complexity exists in the VT task domain.

VT performs the engineering task of designing elevator systems. It must use the customer's functional specifications to select equipment and produce a parts configuration that meets these specifications as well as safety, installation, and maintenance requirements. Because of the large number of potential part combinations and the need for customizing the layout to the space available in individual buildings, VT must construct a solution. Like MOLGEN, VT tries to order its decisions so that they are made only when all relevant constraints are known; it guesses only when stuck.

Unlike MOLGEN, VT's decisions about part selection and placement are so interdependent that plausible reasoning (guessing) is a major feature of its search for a solution. Thus, VT's problem-solving strategy is predominantly one of constructing an approximation and successively refining it.

Systems that use plausible reasoning must be able to identify bad guesses and improve on these decisions in a way which helps converge on a solution. VT is similar to AIR/CYL (Brown 1985) and PRIDE (Mittal and Araya 1986) in that it uses a knowledge-based approach to direct this search; that is, it uses domain-specific knowledge to decide what past decisions to alter and how to alter them. This approach contrasts with EL (Sussman 1977; Stallman and Sussman 1977), an expert system which shares many architectural features with VT but which uses domain-independent strategies to limit the search during the backtracking phase. As with EL, the VT architecture makes clear the role that domain-specific knowledge plays in the system and the interconnections among decisions used to construct and refine a solution. This architecture provides the basis for VT's explanation facility, which is similar to that of EL and the related CONSTRAINTS language (Sussman and Steele 1980), with some extensions. We have exploited the structure provided by this architecture even further by using it to manage VT's knowledge acquisition.

VT's architecture provides structure for a representation of its domain-specific knowledge that reflects the function of the knowledge in problem solving. This representation serves as the basis for an automated knowledge-acquisition tool, SALT (Marcus,

```

Welcome to VT — The Elevator Design Expert System
1. INPUT   Enter contract information
2. RUN     Process the input data
3. SHOW    Display output information
4. EXPLAIN Explain the results of a run
5. SAVE    Save data for the current contract
6. EXIT    End this session with VT

Enter your command [ INPUT ]: <cr>

```

Figure 1. VT's Top Level Menu.

```

INPUT GD DUTY      GR 24364      ADMINISTRATION CENTER
Car:1
1.   Type of loading      PASSENGER
2.   Machine              GEARED
3.   Machine location     OVERHEAD
4.   Power supply         208-3-60
5.   Capacity             3000
6.   Speed                250
7.   Travel               729
8.   Platform width       70
9.   Platform depth       84
10.  Counterweight location REAR
11.  Counterweight safety NO
12.  Compensation specified NO
Action [ EXIT ]:

```

Figure 2. Completed Sample Input Screen.

McDermott, and Wang 1985; Marcus and McDermott 1986, Stout et al. 1987), which has been used to build VT. SALT elicits from experts all the knowledge VT needs in order to design elevators and represents that knowledge in a way which enables VT's problem-solving method to use it. SALT's knowledge representation can also be used to assess the adequacy of the knowledge base for convergence on a solution.

The next section, "What VT Does," presents VT mainly from a user's point of view. "The VT Architecture" describes the VT architecture in detail, with respect to problem-solving, explanation, and knowledge acquisition. "Management of Knowledge-Based Backtracking" describes how SALT's knowledge base analysis supports VT's domain-dependent backtracking. "Comparison to Other Constructive Systems" compares VT to other expert systems that perform design, planning, or scheduling tasks. "VT's Performance" reports some of VT's performance characteristics.

What VT Does

VT is used by Westinghouse Elevator engineers to design elevator systems to customer specifications. VT has enough domain knowledge to perform the design task unaided. VT also has an interactive capability which allows a user to directly influence its decisions.

The Engineer's Task

Westinghouse Elevator design experts receive data collected from several contract documents. These data are transmitted to the engineering operation by the regional sales and installation offices. Three main sources of information exist: (1) customer requirement forms describing the general performance specifications, such as carrying capacity and speed of travel, and some product selections, such as the style of light fixture in the cab; (2) the architectural and structural drawings of the building, indicating such elements as wall-to-wall dimensions in the elevator shaft (hoistway) and locations of rail supports; and (3)

the architectural design drawings of the elevator cabs, entrances, and fixtures. Because all this information is not necessarily available at the start of a contract, the engineer must sometimes produce reasonable guesses for incomplete, inconsistent, or uncertain data to enable order processing to tentatively proceed until customer verification is received. (These guesses are in addition to whatever guesses might be required during a problem-solving episode based on these data.)

Given this information, experts attempt to optimally select the equipment necessary and design its layout in the hoistway to meet engineering, safety code, and system performance requirements. This task is a highly constrained one. A completed elevator system must satisfy constraints such as the following: (1) there must be at least an 8-inch clearance between the side of the platform and a hoistway wall and at least 7 inches between the platform side and a rail separating two cars; (2) a model 18 machine can only be used with a 15, 20, or 25 horsepower motor; and (3) the counterweight must be close enough to the platform to provide adequate traction but far enough away to prevent collision with either the platform or the rear hoistway wall (by an amount dependent on the distance of travel).

The design task also encompasses the calculation of the building load data required by the building's structural engineers, the reporting of the engineering and ordering data required for the field installation department and regional safety code authorities, and the reporting of the mechanical manufacturing order information.

A Quick Look at VT in Action

VT is comprised of several distinct parts, described briefly in the sample interactions which follow. VT prompts appear in boldface. User replies appear in bold italics.

Figure 1 illustrates the top menu, where the user indicates what VT is to do. The INPUT command allows the user either to enter data on a new job or to modify data from an existing job. The other modes use previously input data. VT displays a default command in brackets at the bottom of the

screen that the user can issue by hitting a carriage return (<cr>). Users can also issue single or multiple commands by typing only a portion of a command word or the number in front of it.

VT's input is menu driven, allowing entire screens of questions to be answered at once by providing defaults wherever possible. The input mode also provides consistency checking of data and a general question-asking mechanism that is used throughout VT. A completed sample input screen is shown in figure 2. Prompts for data appear on the left, defaults and input on the right.

Using a simple command language, the user can confirm some or all values shown, enter or modify values, or register uncertainty about values. Fourteen of these data menus currently exist in the INPUT portion of VT. Once all the data have been entered, the user returns to the top menu, at which point the data can be saved for future use (SAVE) or used immediately in the design task (RUN).

As VT runs, it tentatively constructs an elevator system by proposing component selections and relationships. At the same time, VT specifies constraints with which to test the acceptability of the resulting design and tests each constraint whenever enough is known about the design to evaluate it. Whenever constraints are violated, VT attempts to alter the design (for example, by selecting more expensive equipment) in order to resolve the problem. We refer to these alterations as fixes. VT reports any such constraint violation and the fix that is made, as in figure 3.

There are two types of fix reports. The report shown for MAXIMUM-TRACTION-RATIO is the more common version. It mentions the constraint that was violated, describes the degree of the violation, and lists the corrective action taken. The fix report describing the change to CAR-RUNBY is a special case. This version is used when VT makes an initial estimate for a value in order to calculate a precise value for it. The value of the constraint is the precise value; the estimate is simply changed to this value.

During a noninteractive run, VT

The CAR-RUNBY (estimated to be 6) has been changed to 6.125.
The MACHINE-SHEAVE-HEIGHT (estimated to be 30) has been changed to 26.
The CWT-STACK-WEIGHT (estimated to be 4316.25) has been changed to 4287.36.
The MAXIMUM-TRACTION-RATIO constraint was violated. The TRACTION-RATIO was 1.806591, but had to be ≤ 1.783873 . The gap of 0.2272000E-01 was eliminated by the following action(s):
Decreasing CWT-TO-PLATFORM-FRONT from 4.75 to 2.25
Upgrading COMP-CABLE-UNIT-WEIGHT from 0 to 0.5000000E-01
The MINIMUM-MAX-CAR-RAIL-LOAD constraint was violated. The MAX-CAR-RAIL-LOAD was 6000, but had to be ≥ 6722.295 . The gap of 722.3 was eliminated by the following action(s):
Upgrading CAR-RAIL-UNIT-WEIGHT from 11 to 16
The MINIMUM-PLATFORM-TO-CLEAR-HOISTWAY-RIGHT constraint was violated. The PLATFORM-TO-CLEAR-HOISTWAY-RIGHT was 7.5, but had to be ≥ 8 . The gap of 0.5 was eliminated by the following action(s):
Decreasing CAR-RETURN-RIGHT from 3 to 2.5
The MINIMUM-PLATFORM-TO-CLEAR-HOISTWAY-LEFT constraint was violated. The PLATFORM-TO-CLEAR-HOISTWAY-LEFT was 7.5, but had to be ≥ 8 . The gap of 0.5 was eliminated by the following action(s):
Decreasing CAR-RETURN-LEFT from 25.5 to 25
The MAXIMUM-MACHINE-GROOVE-PRESSURE constraint was violated. The MACHINE-GROOVE-PRESSURE was 149.5444, but had to be ≤ 119 . The gap of 30.544 was eliminated by the following action(s):
Increasing HOIST-CABLE-QUANTITY from 3 to 4
The MINIMUM-HOIST-CABLE-SAFETY-FACTOR constraint was violated. The HOIST-CABLE-SAFETY-FACTOR was 8.395078, but had to be ≥ 10 . The gap of 1.60492 was eliminated by the following action(s):
Upgrading HOIST-CABLE-DIAMETER from 0.5 to 0.625
The MINIMUM-MACHINE-BEAM-SECTION-MODULUS constraint was violated. The MACHINE-BEAM-SECTION-MODULUS was 24.7, but had to be ≥ 24.87352 . The gap of 0.1735 was eliminated by the following action(s):
Upgrading MACHINE-BEAM-MODEL from S10X25.4 to S10X35.0
The CHOICE-SET-HOIST-CABLE-DIAMETER constraint was violated. The HOIST-CABLE-DIAMETER was 0.625, but was constrained to be 0.5. The HOIST-CABLE-DIAMETER became a member of the set by the following action(s):
Upgrading MACHINE-MODEL from 28 to 38

Figure 3. Constraint Violation and Fix Report.

uses its own knowledge base to decide how to remedy constraint violations. This knowledge base represents engineering practices that Westinghouse plans to make standard. The RUN can also be done interactively, in which case VT asks for confirmation of each fix before it is actually implemented. If a particular fix is rejected by the user, VT can either find another fix or provide a list of all possible fixes and ask the user to suggest a particular one. Records are kept of user overrides. These overrides are taken into

consideration by the system maintainers when modifying the knowledge base. The overriding of a VT-proposed fix by the user might indicate that a standard does not yet exist on a decision VT makes. It might also be the result of outside factors that were too transitory to make it into the VT knowledge or data base, such as a temporary surplus or a shortage of a particular equipment model.

On completion of the run, control returns to the top menu, at which point the user normally goes into

SHOW LAYOUT SPECS GR 24364	ADMINISTRATION CENTER
Loading: PASSENGER	Governor: B5B Support: STEEL
Capacity: 3000	Governor Cable: 0.375
	Length: 2130
Speed: 250	Hoist Cables: (3)-0.5
	Length: 1089
Operation: 1C-2BC-ERL	Compensation: 3/16-CHAIN
	Length: 993
Travel: 729	Car sling: 2.5B-18
Stops: 6	Crosshead Beam: W8X18
Openings: 6	Platform Thickness: 6.625
Machine: 28	Sling Weight..... 292
Sheave: 30	Platform Weight..... 738
Deflector Sheave: 20	Safety Weight..... 465
Groove: K3269 Pressure: 90.03	Cab Weight.....1668
Angle of Contact: 159.09	Misc. Weight..... 434
Traction Ratio: 1.79	Total Car Weight.....3609
Machine Load: 11691	Counterweight Weight: 4824
Motor H.P.: 20	Subweight Weight:
Power Source: —	Buffer Reaction Car: 26437
Power Supply: 208-3-60	Cwt: 19296
4287 Rails.....Car: 16 Cwt: 11	Machine Weight: 1700
	Heat Emission in M.R.: —
Guide Shoes..Car: 6-R Cwt: 3-R	Cable Hanger —
Buffer.....Car: OH-1 Cwt: OH-1	Safety to Pit: 42
Stroke.....Car: 8.25 Cwt: 8.25	
Safety.....Car: B1 Cwt: —	
Press RETURN to continue [MENU]: show layout cwt	

Figure 4. Show Screen for Layout Specs.

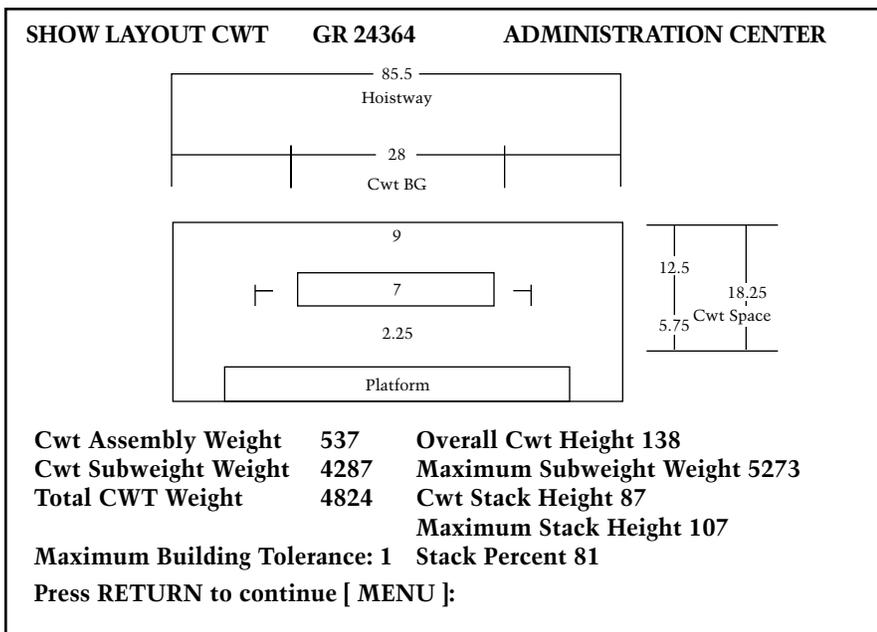


Figure 5. Show Screen (Layout CWT).

SHOW mode. SHOW allows users to view data a screenful at a time. Some of the screens are intended for just such a review, and others are intended

as input data for other Westinghouse systems (such as manufacturing-oriented programs, cost estimators, and a computer-aided drawing system). Fig-

ures 4 and 5 are representative of the sixteen SHOW screens that currently exist; the user accesses these screens by a tree of menus similar to the input menu.

If the user sees something unusual while in SHOW (for example, an unexpected value), the EXPLAIN mode can be used to determine the cause. EXPLAIN can also be used by relative novices to understand how VT performs the design task.

The user interacts with VT's explanation facility by asking questions. The type of information given in the explanation depends on the type of question asked. VT's explanation facility currently provides several types of queries that can be asked about individual system values. These query types are discussed in detail in the next section. The sample interaction in figure 6 demonstrates some of the tools the explanation facility provides, including the use of VT's lexicon of synonyms for system value names.

The only major part of VT that is not visible in figures 1-6 is VT's database. The database is read only and primarily contains data about pieces of equipment and machinery that VT must configure. Each piece of equipment has its own table; the rows of each of these tables represent different models of the equipment from which to choose, and the columns represent attributes relevant to the type of equipment. These attributes can be restrictions on each model's use (for example, maximum elevator speed or maximum load supported by the equipment), values of equipment attributes (for example, height and weight), or lists of model numbers of compatible pieces of equipment.

Calls to the database indicate which table is to be used and what value is to be returned. This value can be either the name of the particular model or the value of one of its attributes. A call might also include an arbitrary number of constraints on the values of each column.

In the event that multiple entries in the database satisfy all the constraints in a call, each table is ordered along an equipment attribute (for example, size) to indicate a preference or priority. The entries in a table are examined

from best to worst, and the first entry satisfying all the constraints is the one from which the return value is obtained.

The VT Architecture

VT solves its problem by constructing an approximate elevator design and successively refining it. The process of constructing an approximate design is forward chaining. Each step in this phase extends the design by procedures that use input data or results of prior decisions to determine a value for a design parameter. Some of these steps embody heuristic knowledge about how to propose an approximate design extension. These steps are needed when the decision is under-constrained or when it must be based on partial information. As VT builds a proposed design, constraints on the elevator system are specified whenever enough information is available to determine their values. The control in this constructive phase is data driven; any step can be taken as soon as the information called for by the procedure associated with the step is available. As it extends the design, VT also builds a dependency network that records for each value which other values were used to obtain it.

The dependency network developed during the forward-chaining constructive phase is enough to identify all contributors to a violated constraint and the value it constrains. These contributors represent potential points to backtrack to in order to revise the proposed design. However, domain expertise is needed to indicate what changes in the proposed design are least costly in real-world terms. Although it is not possible to assign a dollar cost to each revision, domain knowledge determines which of the potential alterations are legal as well as the order of preference among the legal ones.

Demons are used to check for constraint violations; whenever enough is known about the proposed design to supply values for both a constraint and the value it constrains, they are compared. Whenever VT detects a constraint violation, it tests the effectiveness of suggested changes in order of decreasing preference rating. As VT

EXPLAIN GR 24364 ADMINISTRATION CENTER

Explain: *how car runby*

The CAR-RUNBY was determined by a fix.

The CHOICE-SET-CAR-RUNBY constraint was violated.

The CAR-RUNBY was 6, but was constrained to be 6.125.

The CAR-RUNBY was changed from 6 to 6.125.

How [CHOICE-SET-CAR-RUNBY]: <cr>

The CHOICE-SET-CAR-RUNBY (6.125) = PIT-DEPTH (72) - [PLATFORM-HEIGHT (6.625) + SAFETY-HEIGHT (9) + CAR-BUFFER-HEIGHT (28.75) + CAR-FOOTING-CHANNEL-HEIGHT (3.5) + CAR-BUFFER-BLOCKING-HEIGHT (18)]

How [PIT-DEPTH]: <cr>

The PIT-DEPTH (72) was input by Bob Roche on 25-MAR-1985.

How [PLATFORM-WEIGHT]: *safety height*

The SAFETY-HEIGHT (9) was determined by a database lookup.

It was found in the HEIGHT column of the SAFETY table.

It met the following constraints:

MODEL = SAFETY-MODEL (B1)

How [SAFETY MODEL]: <cr>

The SAFETY-MODEL (B1) was determined by a database lookup.

It was found in the MODEL column of the SAFETY table.

It had the SMALLEST HEIGHT that met the following constraints:

MAX-SPEED > SPEED (250)

MAX-PLATFORM-WIDTH >= PLATFORM-WIDTH (70)

MIN-PLATFORM-WIDTH <= PLATFORM-WIDTH (70)

How [SPEED]: *what if safety model B4*

The SAFETY-MODEL is currently B1.

If it were B4, the following major changes would occur:

NAME:	ACTUAL:	PROPOSED:
MACHINE-GROOVE-PRESSURE	114.118	155.563.
TRACTION-RATIO	1.80679	1.76682.
CWT-OVERTRAVEL	49.835	52.835.
CAR-BUFFER-REACTION	26709.4	27652.4.
CWT-STACK-PERCENT	84.1122	88.148.
CWT-BUFFER-REACTION	19684	20627.0.
CWT-PLATE-QUANTITY	90	94.3184.
CWT-WEIGHT	4921.0	5156.76.
CAR-BUFFER-LOAD	6677.35	6913.11.
CAR-WEIGHT	3677.35	3913.11.
DEFLECTOR-SHEAVE-DIAMETER	25	20.
CAR-BUFFER-BLOCKING-HEIGHT	18	17.125.
HOIST-CABLE-MODEL	(4)-0.5	(3)-0.5.
CAR-RUNBY	6.125	6.
SAFETY-MODEL	B1	B4.

Would you like to see ALL values which would change [NO]: <cr>

Would you like to implement this [NO]: <cr>

How [MACHINE-GROOVE-PRESSURE]: *safety load*

There is more than one SAFETY-LOAD:

1. SAFETY-LOAD-CAR-SIDE-CAR-TOP
2. SAFETY-LOAD-CAR-SIDE-CAR-BOTTOM
3. SAFETY-LOAD-CWT-SIDE-CAR-TOP
4. SAFETY-LOAD-CWT-SIDE-CAR-BOTTOM

Which would you like to know about?

[SAFETY-LOAD-CAR-SIDE-CAR-TOP]: 2

Figure 6. A Sample Interaction with the Explanation Facility.

(1) MACHINE-MODEL step:

IF a value has been generated for SUSPENDED-LOAD, and there is no value for MACHINE-MODEL,

THEN look in the database in the MACHINE table for the entry with the SMALLEST WEIGHT whose listing for MAX-LOAD is greater than the SUSPENDED-LOAD.

Retrieve the value under MODEL for that entry and assign that value to MACHINE-MODEL.

Leave a trace that SUSPENDED-LOAD contributed to MACHINE-MODEL.

Leave a declarative representation of the details of the database call.

Figure 7. Machine-Model Step.

moves through the list of potential fixes for a constraint violation, it first tries every individual fix at a given preference level. Next it tries to combine each fix at the current preference level with those of greater or equal preference.

Once VT identifies a change to explore, it first verifies that no constraints on the changed value itself are violated by the change. It then makes the proposed change and works through the implications according to its knowledge about constructing a proposed design. (Constraints can be numeric or symbolic, and procedures for determining values often involve nonlinear functions such as selections from the database.) VT continues this procedure until it has enough knowledge to evaluate the originally violated constraint. If a proposed change violates the constraints, it is rejected, and another selection is made. This lookahead is limited because it only considers constraints on the changed value and the originally violated constraint. The purpose of this lookahead is to limit the work done in exploring the implications of a proposed guess until VT has reason to believe it is a good guess. Once a good guess has been identified, VT applies a truth maintenance system; that is, it uses the dependency network constructed during the forward-chaining phase to identify and remove any values that might be inconsistent with the changed value. VT then reenters the data-driven constructive phase for extending the design with the new data.

A Detailed Look at Problem Solving

In order to better illustrate how VT

soon as a value for SUSPENDED-LOAD is made available; then uses this value to supply MACHINE-MODEL. Leaving a trace of the contribution adds to the dependency network used by the truth maintenance system in backtracking. Leaving a declarative representation of the action taken by this rule is used by the explanation facility.

To see how this step might interact with others, consider the two steps shown in figure 8.

According to the control shown in

(2) MACHINE-SHEAVE-DIAMETER step:

IF a value has been generated for MACHINE-MODEL, and there is no value for MACHINE-SHEAVE-DIAMETER,

THEN look in the database in the MACHINE table for the entry whose listing for MODEL is the same as MACHINE-MODEL.

Retrieve the value under SHEAVE-DIAMETER for that entry and assign that value to MACHINE-SHEAVE-DIAMETER.

Leave a trace that MACHINE-MODEL contributed to MACHINE-SHEAVE-DIAMETER.

Leave a declarative representation of the details of the database call.

(3) MACHINE-GROOVE-PRESSURE-FACTOR step:

IF a value has been generated for HOIST-CABLE-DIAMETER, and there is no value for MACHINE-GROOVE-PRESSURE-FACTOR,

THEN compute $2 * \text{HOIST-CABLE-DIAMETER}$.

Assign the result to MACHINE-GROOVE-PRESSURE-FACTOR.

Leave a trace that HOIST-CABLE-DIAMETER contributed to MACHINE-GROOVE-PRESSURE-FACTOR.

Leave a declarative representation of the details of the calculation.

Figure 8. Sheave-Diameter and Pressure-Factor Steps.

arrives at a solution, we describe the forwardchaining and backtracking done in a small portion of the sample run. The detail focuses on steps leading to the specification of MACHINE-GROOVE-PRESSURE and its constraint MAXIMUM-MACHINE-GROOVE-PRESSURE and follows the backtracking initiated by a violation of this constraint.

A step to extend the proposed design specifies a value for a design parameter, often using results of decisions already made. For example, the step to select the model of the machine that moves the elevator car can be given the English translation shown in figure 7.

The first line of this step specification sets up the forward-chaining control. This rule is eligible to fire as

figures 7 and 8, step 1 must be applied before step 2 because step 1 creates the conditions under which step 2 is satisfied. If step 3 is satisfied at the same time as either of the other steps, it does not matter which procedure is applied first.

The machine moves the elevator by turning the machine sheave. The machine sheave contains grooves that grip the hoist cables which support the elevator car. Some pressure is required, but if the pressure on each individual cable is too great, there is excessive wear on the cables. Steps 1 and 2 are on the inference chain that produces a value for MACHINE-GROOVE-PRESSURE. This value is the result of a calculation using MAX-TOTAL-LOAD-CAR-SIDE, MACHINE-SHEAVE-DIAMETER, and

HOIST-CABLE-QUANTITY. Step 3 is on the inference chain that produces a value for MAXIMUM-MACHINE-GROOVE-PRESSURE. This value is a function of the MACHINE-GROOVE-MODEL, the SPEED the elevator will travel, and MACHINE-GROOVE-PRESSURE-FACTOR. Once values for both MACHINE-GROOVE-PRESSURE and MAXIMUM-MACHINE-GROOVE-PRESSURE are available, they are compared. Because the constraint is a maximum, the constraint is flagged as violated if the value of MACHINE-GROOVE-PRESSURE is greater than the value of MAXIMUM-MACHINE-GROOVE-PRESSURE. Flagging the constraint as violated causes VT to shift control into fix exploration.

As a first step in exploring remedies for the constraint violation, VT proposes potential remedies. For this particular violation, a propose-fix step for the VT knowledge base looks as shown in figure 9. (This is an abbreviated listing of fixes for MAXIMUM-MACHINE-GROOVE-PRESSURE. We return to a complete treatment of this example in "Management of Knowledge-Based Backtracking.")

Downgrading the MACHINE-GROOVE-MODEL to one that grips the cable less increases the allowable MAXIMUM-MACHINE-GROOVE-PRESSURE. Increasing the HOIST-CABLE-QUANTITY distributes the load and decreases the actual MACHINE-GROOVE-PRESSURE on each groove. VT's domain expert felt these two potential fixes would be practical to attempt. Of the two fixes, the first is preferable.

VT first considers a downgrade of MACHINE-GROOVE-MODEL by trying to select the next higher groove according to the preference ordering.¹ If there is such a preferred groove, VT determines what the MAXIMUM-MACHINE-GROOVE-PRESSURE for this groove is. If this value is not less than the value of MACHINE-GROOVE-PRESSURE, VT tries to downgrade the groove model further. When there are no longer any models to try (there are only two groove models), VT considers an increase of HOIST-CABLE-QUANTITY by adding 1 to its current value. It first checks to see whether this quantity is

**IF there has been a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE, THEN try a DOWNGRADE for MACHINE-GROOVE-MODEL which has a preference rating of 1 because it CAUSES NO PROBLEM.
Try an INCREASE BY-STEP of 1 of HOIST-CABLE-QUANTITY which has a preference rating of 4 because it CHANGES MINOR EQUIPMENT SIZING.**

Figure 9. A Propose-Fix Step.

larger than the MAXIMUM-HOIST-CABLE-QUANTITY (which in any application is never more than six cables). If not, VT then recomputes the MACHINE-GROOVE-PRESSURE using the new HOIST-CABLE-QUANTITY to see if this quantity brings the pressure under the maximum. If it does not, VT tries adding another hoist cable and repeats the procedure. If VT exceeds the MAXIMUM-HOIST-CABLE-QUANTITY before bringing MACHINE-GROOVE-PRESSURE under its maximum, it then attempts a combination of the two fixes. If none of the specified fixes resolve the violation, VT has reached a dead end (that is, the constraint violation cannot be corrected). In the sample run shown previously, the proposed design already employed the preferred groove at the time of the constraint violation; adding a single hoist cable was the selected remedy.

Once VT finds the fix it wants to implement, it uses the dependency network built during the forward chaining to remove any values that depended on the one it changed. It then returns to the forward-chaining phase with the new HOIST-CABLE-QUANTITY and continues.

A Detailed Look at the Explanation Facility

Every decision VT makes must be justifiable to the user. This condition is provided for by making a record of each decision as it is made. The dependency network built for VT's truth maintenance system can provide the foundation for a very useful explanation facility (Doyle 1979; Sussman and Steele 1980). This network is augmented by the details of the contribution relation, for example, a description of an algebraic formula or the relation between values required by a precondition. In addition, VT records adjustments to the proposed

design that it makes, such as fixes of constraint violations. The explanation facility pieces these individual actions together to describe VT's line of reasoning.

VT's explanation facility does more than just examine past decisions; it also performs some hypothetical reasoning to demonstrate the effect of alternative decisions the user suggests. Hypothetical explanations are relatively simple to construct given the VT knowledge representation. What the system must do in order to answer hypothetical queries is closely related to how it resolves constraint violations.

Explaining Past Decisions. The how query is probably the most fundamental and can be thought of as asking the question "How did you determine the value of <x>?" First, the explanation facility looks for the appropriate node in the dependency network that recorded the decision which VT made regarding the value assigned to <x>. This decision record includes, for example, not only a formula but also any conditions in the system that made the formula appropriate. The dependency network provides pointers to the actual values that were used in determining the value in question.

If the user were to ask how the machine groove pressure was determined, VT would respond with something like the following:

The MACHINE-GROOVE-PRESSURE (90.0307) = MAX-TOTAL-LOAD-CAR-SIDE (6752.3042) / [[MACHINE-SHEAVE-DIAMETER (30) * 0.5] * HOIST-CABLE-QUANTITY (5)]

The machine groove pressure was determined by a calculation, which is displayed in terms of the names of the system values and their values.

If the value being explained was obtained via a database lookup, the

explanation facility responds with something like the following:

**The MOTOR-MODEL (20HP) was determined by a database lookup. It was found in the MODEL column of the MOTOR table. It had the SMALLEST HORSEPOWER that met the following constraints:
HORSEPOWER > REQUIRED-MOTOR-HP (18.705574)**

The facility reports the name of the table and the column within the table from which the value was obtained as well as what criterion was used in ordering the table. It then lists the constraints that were applied to the attributes in the table which narrowed the choice.

If the method used to calculate the value in question was selected according to a precondition, the description of the method is followed by a description of the precondition, as follows:

**The CAR-RETURN-LEFT (25) = PLATFORM-WIDTH (70) - [OPENING-WIDTH-FRONT (42) + CAR-RETURN-RIGHT (3)]
This particular method was used because:
[DOOR-SPEED-FRONT = TWO]
AND [OPENING-STRIKE-SIDE-FRONT = RIGHT]**

In addition, the how query finds possible reasons why a quantity in the system might have a value that the expert believes to be out of the ordinary, unexpected, or just plain incorrect. In VT, several kinds of "unusual" values can occur, as the following paragraphs illustrate:

Conflicting input values: Some inputs to VT can come from multiple sources. If these sources specify different values, one is chosen (by applying a specified strategy), and a record is made of the event. Obviously, the choice can be incorrect, which can cause unusual values to propagate throughout the system.

Inconsistent input values: This situation occurs when two input values violate an expected relationship between them. For example, inputs exist for the number of front openings, number of rear openings, and the total number of openings in an elevator shaft. Obviously, "front" plus

"rear" should equal "total," but if such is not the case, a decision is made about how to make the values consistent, and a record is made of the event.

Unusual input values: Some inputs have a reasonable range of values specified. A value outside the reasonable range is allowed (as long as it does not violate the absolute range)

cess of extending a design to say that a value is determined by its direct contributors or unusual decisions which directly change its value. Everything upstream in the dependency network contributes to the proposed value. The explanation facility allows the user to step back through the network by repeated questioning and provides default queries after each

Explain: how traction ratio

**The TRACTION-RATIO (1.796574) =
MAX [TRACTION-RATIO-CAR-TOP-FULL (1.759741)
TRACTION-RATIO-CAR-BOTTOM-FULL (1.796574)
TRACTION-RATIO-CAR-TOP-EMPTY (1.742178)
TRACTION-RATIO-CAR-BOTTOM-EMPTY (1.696701)]**

The value for TRACTION-RATIO may be unusual because:

- (1) The MACHINE-MODEL was changed due to a constraint on the HOIST-CABLE-DIAMETER. (Depth = 3)**
- (2) The CAPACITY was an inconsistent input value. (Depth = 3)**

Figure 10. An Explanation Noting Unusual Contributors.

but is an indication that VT is receiving an input which is out of the ordinary. As stated earlier, this unusual value can propagate other unusual values throughout the system.

Default input values: If the user chooses not to answer a particular question in the input, a default value is assigned. The chances that the default chosen is actually the correct value depends on the particular question.

Fixed values: A value changed by the fix mechanism can look unusual to a user, particularly if the value changed is an input or if a low-preference fix was required.

When the user makes a how query about a value, unusual occurrences are reported as well:

Explain: how hoist cable quantity

The HOIST-CABLE-QUANTITY (4) was determined by a fix:

The MAXIMUM-MACHINE-GROOVE-PRESSURE constraint was violated. The MACHINE-GROOVE-PRESSURE was 149.5444, but had to be <= 119.

The gap of 30.544 was eliminated by the following action(s):

Increasing HOIST-CABLE-QUANTITY from 3 to 4

Of course, it is simplifying the pro-

answer to aid in this process, as shown earlier in "What VT Does." The facility also searches the upstream network on its own and in answering any how query reports any unusual decisions made about upstream contributors. In searching for reasons why <x> might be unusual, the explanation facility examines all the items that directly contributed to <x> as well as the items used in evaluating any preconditions on <x>'s method. This examination is recursive in that each of these contributors is also examined similarly and so on until the explanation facility grounds out on either inputs or constants.

Figure 10 illustrates an unusual explanation; the user asks how TRACTION-RATIO was determined. The depth indicates how far upstream the contributor is.

Hypothetical Reasoning. The data-driven control for the forward-chaining construction of the proposed design assumes that the dependency network built while the design was extended is a directed acyclic graph. Because of this assumption, hypothetical queries can proceed in two directions—upstream and downstream. The two hypothetical query types—why not and what if—differ in their emphasis on what direction is of

Explain: why not safety model B4

The SAFETY-MODEL (currently B1) could be B4, but that is less desirable because it has a larger HEIGHT. A SAFETY-MODEL of B1 was selected because it met the following constraints:

Its MAX-SPEED (500) was at least as much as the SPEED (250).

Its MAX-PLATFORM-WIDTH (93) was not less than the PLATFORM-WIDTH (70).

Its MIN-PLATFORM-WIDTH (54) was not more than the PLATFORM-WIDTH (70).

Figure 11. Why Not Explanation

interest to the user. Thus, the answer to the query is reported differently depending on the query type. However, fixes for constraint violations can form loops in VT's line of reasoning. Downstream constraint violations can cause upstream design adjustments that can affect the node from which the query originated. Thus, when hypothesizing about a change to a node in the dependency network, the system must be run to quiescence to ensure that the reported causes or effects are taken from a consistent, acceptable design.

The why not query can be thought of as asking the question "Why wasn't the value of <x> a particular value?" This question is appropriate if the user expected (or desired) a certain

value, and VT did not produce it. The explanation facility then suggests what has to be done in order to obtain the desired result. The how query does a search for reasons why a value might be unexpected, and the why not query looks for a way to bridge the gap between the system's model and that of the user.

If the user expected VT to choose a larger safety, the question "why not safety model B4" could be posed. The results are shown in Figure 11.

Thus, in this case, the user's expectation is possible but not preferred. Here, the explanation facility locates all constraints in the system that constrained the safety model (including implicit constraints in database calls) and reports them.

The following case is the opposite. The suggested value is preferred but is not possible, except perhaps by changing values upstream (for example, introducing nonpreferred values elsewhere).

Explain: why not safety model B1

A SAFETY-MODEL of B1 would have been used (instead of B4) if: The PLATFORM-WIDTH were 84 instead of 86.

In order to handle this second case, VT uses knowledge that was acquired solely for the purpose of handling hypothetical queries about the value of SAFETY-MODEL. The form of the knowledge required is the same as that required for fixing designs which violate constraints. VT must have knowledge of what contributors to SAFETY-MODEL are changeable, the relative preference for possible changes, and the nature of the change in a contributor that would produce the desired difference in SAFETY-MODEL. As mentioned earlier, the system continues to completion to verify that changes made to produce the desired SAFETY-MODEL can stay in place regardless of any fixes for subsequent constraint violations. If the proposed changes cannot be incorporated into an acceptable design—that is, some constraint violation is impossible to fix—this condition is reported. Otherwise, the explanation facility is poised to describe the effects of these changes in the same way it does for what if queries, and VT offers to display this information to the user.

The what if query can be thought of as asking the question "What would happen if I changed <x> to be a particular value?" The user then sees the impact this change would make on the system when VT lists which important system values would change. (The term important is predefined and is part of VT's knowledge base.) Sixty system values are currently considered important in this context, but usually only a relatively small subset of these 60 change in a given scenario; thus, the user is not overwhelmed by information. Figure 12 shows the what if explanation of the scenario that was shown in figure 11.

If the user does wish to examine

Explain: what if safety model B4

The SAFETY-MODEL is currently B1.

If it were B4, the following major changes would occur:

NAME:	ACTUAL:	PROPOSED:
MACHINE-GROOVE-PRESSURE	114.118	155.563
TRACTION-RATIO	1.80679	1.76682
CWT-OVERTRAVEL	49.835	52.835
CAR-BUFFER-REACTION	26709.4	27652.4
CWT-STACK-PERCENT	84.1122	88.148
CWT-BUFFER-REACTION	19684	20627.0
CWT-PLATE-QUANTITY	90	94.3184
CWT-WEIGHT	4921.0	5156.76
CAR-BUFFER-LOAD	6677.35	6913.11
CAR-WEIGHT	3677.35	3913.11
DEFLECTOR-SHEAVE-DIAMETER	25	20
CAR-BUFFER-BLOCKING-HEIGHT	18	17.125
HOIST-CABLE-MODEL	(4).5	(3).5
CAR-RUNBY	6.125	6
SAFETY-MODEL	B1	B4

Would you like to see ALL values which would change [NO]: <cr>

Would you like to implement this [NO]:

Figure 12. What If Explanation.

detailed information, the option is provided to see all the values that would change. The ability to implement a suggested change is provided. As was the case with the fix mechanism when run interactively, this option is provided as a way to force VT to produce nonstandard results (perhaps in response to inventory fluctuations or other transient situations).

Internally, the why not and what if queries are virtually identical. Because they both propose a value for a particular quantity, they must be able to go upstream and modify values in order to make the system consistent with the new value and then propagate the value downstream. This process is exactly what the fix mechanism follows, and in fact, these two queries effectively add a dynamic constraint to the system. As mentioned earlier, VT must have fix knowledge to go with these constraints, something which is impractical for all values that VT derives while it constructs a design. When the user asks a why not or what if query about a value that VT has no fix knowledge for, the user is so warned. The what if report might still be of interest, but it is then up to the user to verify upstream consistency.

SALT: A Look at Knowledge Acquisition

VT's problem-solving strategy imposes an organization on the system's knowledge that can be exploited for knowledge acquisition. Given the assumed propose-and-revise strategy, domain-specific knowledge must perform one of three roles with respect to the problem solver: (1) PROPOSE-A-DESIGN-EXTENSION, (2) IDENTIFY-A-CONSTRAINT on a design extension, or (3) PROPOSE-A-FIX for a constraint violation. A representation scheme for a domain-specific knowledge base such as VT's should recognize these roles and the interdependencies among them. Understanding knowledge roles and relationships is crucial to acquisition and maintenance of the knowledge base and provides the key to how and when the knowledge should be used by the problem solver.

SALT is an automated knowledge-

acquisition tool that assumes the systems it generates will use a propose-and-revise problem-solving strategy. SALT acquires knowledge from an expert and generates a domain-specific knowledge base compiled into rules. This compiled knowledge base is then combined with a problem-solving shell to create an expert system. SALT maintains a permanent, declarative store of the knowledge base which is updated during interviews with the domain expert and which is the input to the compiler. This intermediate representation language seeks to make the function of domain knowledge explicit.

As with CONSTRAINTS, SALT's representation scheme is built around the framework of a dependency network. For SALT, each node in the network is the name of a value; this name can be that of an input, a design parameter, or a constraint. Three kinds of directed links represent relations between nodes: (1) "contributes-to" links A to B if the value of A is used in a procedure to specify a value for B; (2) "constrains" links A to B if A is the name of a constraint, B is the name of a design parameter, and the value of A places some restriction on the value of B; and (3) "suggests-revision-of" links A to B if A is the name of a constraint, and a violation of A suggests a change to the currently proposed value of B. Each of these links is supported by additional information in the knowledge base: (1) contributes-to links are supported by details of how contributors are combined to specify the value of the node pointed to; (2) constrains links are supported by a specification of the nature of the restriction; and (3) suggests-revision-of links are supported by a declaration of the nature of the proposed revision (for example, direction and amount of change) and its relative preference.

For SALT, the knowledge-acquisition task becomes one of fleshing out the knowledge base using these representational primitives. SALT allows users to enter knowledge piecemeal starting at any point. The grain size of the pieces corresponds roughly to the three knowledge roles for the propose-and-revise strategy: Users can supply a procedure for specifying a parameter

value, identify a constraint on a parameter value, or suggest a remedy for a constraint violation. SALT keeps track of how the pieces are fitting together and warns the user of places where pieces might be missing or creating inconsistencies.

SALT users must first specify which of the three roles each piece of entered knowledge plays. Once this choice is made, SALT presents a set of prompts for the detailed knowledge required by this role. For example, a filled-in schema for PROPOSE-A-DESIGN-EXTENSION for CAR-RETURN-LEFT is shown in figure 13; where SALT prompts appear on the left and user responses on the right.

The IDENTIFY-A-CONSTRAINT schema prompts for similar information to acquire a procedure for determining a value (or values in the case of a set constraint) for the constraint. In addition, the schema requires the user to specify what parameter is constrained and what kind of constraint it is (for example, a maximum).

Collection of information to direct backtracking is also highly structured. Each piece of PROPOSE-A-FIX knowledge is a proposal for remedying the violation of a particular constraint by changing one of the decisions made while extending a design. Procedures used in the forward-chaining portion of extending a design produce values the expert would prefer in an under-constrained case. Associated with the potential fixes is some reason why they are less preferred than the originally proposed value. The reasons are drawn from the following list:

- 1. Causes no problem**
- 2. Increases maintenance requirements**
- 3. Makes installation difficult**
- 4. Changes minor equipment sizing**
- 5. Violates minor equipment constraint**
- 6. Changes minor contract specifications**
- 7. Requires special part design**
- 8. Changes major equipment sizing**
- 9. Changes the building dimensions**
- 10. Changes major contract specifications**
- 11. Increases maintenance costs**
- 12. Compromises system performance**

These effects are ordered from most to least preferred. The reasons mainly reflect concerns for safety and customer satisfaction as well as dollar

cost to the company. Relative position on this scale is significant, but absolute position is not. When more than one fix is suggested to remedy a particular constraint violation, the most preferred fix of those suggested is attempted first.

In addition, the domain expert must indicate the kind of change that should be made. This indication can be a perturbation of whatever the current value is, or it can entail a change that doesn't reference the current value, such as the substitution of some other system value. Figure 14 is an example of a filled-in schema for a fix for MAXIMUM-MACHINE-GROOVE-PRESSURE.

In addition to providing a language for representing domain-specific knowledge, SALT analyzes the knowledge base and guides the user's input to ensure that the knowledge base is complete and consistent. SALT's overall design and operation are described in detail elsewhere (Marcus, McDermott, and Wang 1985; Marcus and McDermott 1986). The next section describes an analysis SALT provides to test any knowledge base it collects for adequacy with respect to the problem-solving method it assumes.

Management of Knowledge-Based Backtracking

The kind of domain-specific information that SALT initially collects to direct backtracking is relatively easy to supply because the expert can focus on one constraint violation at a time. However, a search that relies solely on this local information and ignores potential interactions among fixes for different constraint violations can run into trouble. One naive way to ensure that a system which uses backtracking converges on a solution, if one exists, is to open the search completely and try every possible combination of values for every potential fix before announcing failure. This solution is not practical for domains that have any significant amount of complexity, such as VT's domain. VT can currently encounter 52 different constraint violations. Most constraint violations (37 of 52) have only one fix—one parameter that might be revised. However, typically there are several

Name:	CAR-RETURN-LEFT
Precondition:	[DOOR-SPEED-FRONT = TWO] AND [OPENING-STRIKE-SIDE-FRONT = RIGHT]
Procedure Type:	CALCULATION
Formula:	PLATFORM-WIDTH - OPENING-WIDTH-FRONT + CAR-RETURN-RIGHT

Figure 13. A Completed SALT Schema for a Procedure.

Constraint Name:	MAXIMUM-MACHINE-GROOVE-PRESSURE
Value to Change:	HOIST-CABLE-QUANTITY
Change Type:	INCREASE
Step Type:	BY-STEP
Step Size:	1
Preference Rating:	4
Preference Reason:	CHANGES MINOR EQUIPMENT SIZING

Figure 14. A Completed SALT Schema for a Fix.

or many alternative values a parameter might assume. This case also exists for the remaining constraints with multiple fixes; 10 have two fixes each, 3 have three fixes, and 2 have four potential fixes, with multiple possible instantiations for each fix. A blind search that considered all possible combinations of these fixes would have a potentially large search space. In fact, it might be unnecessarily large because it might not be the case that every fix interacts with every other.

SALT helps manage knowledge-based backtracking by mapping out potential interactions among fixes for different constraint violations. A developer can then examine cases of interacting fixes for their potential to cause trouble for convergence on a solution. Nonproblematic fixes can be handled using local information only. This treatment ignores potential interactions among fixes for different constraints. Trouble spots are treated as special cases that take into account global information.

VT's Local Treatment and Its Trouble Spots

In the local treatment, deciding which upstream value is to be modified is conditioned on individual constraint violations. Potential fixes considered are only those which the domain

expert identified as relevant to the current violation, and these are selected in order of the expert's preference. Until a remedy is found for this violation, all possible combinations of these constraint-specific remedies are tried. If the system reaches a dead end, that is, none of these combinations remedy the local constraint violation, the system announces that there is no possible solution. If fixes for one constraint violation have no effect on other constraint violations, this strategy guarantees that the first solution found is the most preferred and that the system correctly reports failure if no successful fix is found for an individual constraint.

However, it is possible that remedies selected for one constraint violation might aggravate constraint violations that occur further downstream. In some instances, this situation can result in failure to find a solution when one does exist.² In these cases, a fix that appears optimal based on local information would not be preferred if more were known about the search space.

For example, the most preferred fix for one constraint violation might aggravate a downstream constraint violation to such a degree that it reaches a dead end when exploring its own fixes. If less preferred fixes for the first constraint do not have the

same negative effect downstream, then a solution might be possible. The undesired behavior of the system in this case would be a premature announcement of failure.

Another potential problem is that unproductive looping can occur between fixes for two constraint violations if each has a preferred fix with a counteracting effect on the other. This situation occurs, for example, if fixing one constraint violation increases a certain value that leads to the violation of another constraint whose fix results in decreasing the same value, and so on. Repeated violations of the same constraint are not necessarily pernicious, but such a case of antagonistic constraints might result in an infinite loop.

SALT provides a mapping of the interactions among fixes in a knowledge base. It does this mapping using its understanding of dependencies among procedures for extending a design plus identification of constraints and fixes. (See Marcus and McDermott 1986 for a detailed description of this analysis.) We used this map to analyze VT's knowledge base for its potential to get into trouble with a local, constraint-specific search. We then hand coded a special case treatment for the problem spots we found. We plan to automate this entire process in SALT.

VT's Fix Interactions and Their Special Handling

The VT knowledge base contains 37 chains of interacting fixes. Eleven of these chains are short and nonproblematic. The rest represent different entry points for loops on 8 constraints. Two of these looping constraints represent no danger for the local treatment. Three pairs of constraints might cause thrashing under the local treatment and are treated as special cases in VT.

Each of the eleven short chains involve at most three constraints and the effects of only one fix per constraint. The most common scenario for these chains is that when a constraint violation causes one piece of equipment to be upgraded (or increased in size), the values of constraints on related equipment are

affected and might require that the related equipment be upgraded as well. For example, if the number of hoist cables needed for a job exceeds the maximum allowable for the machine model selected, the fix is to choose a larger machine that can accommodate more cables. The machine model's specifications limit what machine sheave heights it can be used with; larger machines require larger machine sheaves. If the current machine sheave is too small for the newly upgraded machine model, a larger machine sheave (the smallest one that meets constraints) is substituted.

The situation involving the two nonproblematic looping constraints, CHOICE-SET-HOIST-CABLE-QUANTITY and CHOICE-SET-HOIST-CABLE-DIAMETER, also involves a rippling effect of upgrading equipment. Most of the equipment selection in VT depends on the weight of other components selected. The hoist cable quantity and diameter depend on hoist cable quantity and diameter (that is, they must be able to support their own weight) as well as properties of other parts that require knowledge of hoist cable quantity and diameter in their selection. The VT strategy estimates the lowest acceptable value for hoist cable quantity and diameter using rough criteria, selects other parts using these estimates, and derives from these estimates a constraint on the quantity and diameter that must be used. If the value of the constraint does not match the initial estimate, quantity and diameter are increased. Violations of other constraints on the system derived from this major equipment selection, such as the MAXIMUM-MACHINE-GROOVE-PRESSURE shown earlier, also call for changing hoist cable quantity or diameter but always in the direction of increasing the values. Furthermore, the VT knowledge base also contains knowledge of MAXIMUM-HOIST-CABLE-QUANTITY and MAXIMUM-HOIST-CABLE-DIAMETER. (SALT asked for this information when fixes were entered that called for increasing the quantity and diameter.) Thus, this loop does not present the danger of infinite looping. Because the values start at

the lowest possible point and always increase until the maximums are reached, the system does not thrash.

Three cases, however, might result in infinite loops under the local treatment. These cases contain a pair of antagonistic constraints that might cause thrashing. A local treatment of one of these constraints, MAXIMUM-MACHINE-GROOVE-PRESSURE, was described earlier. Its antagonistic constraint is MAXIMUM-TRACTION-RATIO. The complete set of potential fixes for each of these is shown in figure 15.

Figure 16 shows the relevant segment of the VT knowledge base as SALT represents it. Constraints are connected to the values they constrain by the dotted arrows at the bottom. Above these arrows is the portion of the dependency network that links the constraint-constrained pairs to their potential fix values. Contributors are linked to the values they contribute to by a solid arrow. In order to make the figure readable, not all contributors are shown. In addition, suggests-revision-of links were omitted. Instead, suggested revisions in response to a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE are surrounded by rectangles, and suggested revisions for violations of MAXIMUM-TRACTION-RATIO are enclosed in ovals.

One scenario can illustrate the potential for thrashing in this part of the network. This scenario uses the knowledge shown in figure 1 plus information supporting the links, including formulas for combining contributors, the nature of constraints, and the suggested direction of revisions. Suppose MAXIMUM-TRACTION-RATIO is violated, and VT responds by increasing CAR-SUPPLEMENT-WEIGHT. This situation increases CAR-WEIGHT, which, in turn, increases SUPPORTED-LOADS. This condition decreases TRACTION-RATIO but increases MACHINE-GROOVE-PRESSURE. An increase in MACHINE-GROOVE-PRESSURE makes it likely for it to exceed its maximum. A violation of MAXIMUM-MACHINE-GROOVE-PRESSURE could call for a decrease of COMP-CABLE-UNIT-WEIGHT, which, in turn, would decrease

COMP-CABLE-WEIGHT, CABLE-WEIGHT, and SUPPORTED-LOADS. Decreasing SUPPORTED-LOADS increases TRACTION-RATIO, making it more likely to violate MAXIMUM-TRACTION-RATIO. At this point, the scenario could repeat itself.

SALT analyzes the knowledge base for scenarios such as this one and produces messages such as the one shown in figure 17.

The top leftmost constraint, MAXIMUM-TRACTION-RATIO, in figure 17 is an arbitrary starting point. Potential fixes for its violation appear in parentheses and indented one level. The suggested changes to three of these values—MACHINE-GROOVE-MODEL, COMP-CABLE-UNIT-WEIGHT, and CAR-SUPPLEMENT-WEIGHT—would make violation of MAXIMUM-MACHINE-GROOVE-PRESSURE more likely, as indicated by its appearance indented below these fixes. Violation of MAXIMUM-MACHINE-GROOVE-PRESSURE, in turn, could call for changes to these same three fix values. The LOOP flags indicate that these changes might make a violation of MAXIMUM-TRACTION-RATIO more likely. As shown by a lack of nesting, decreasing the CWT-TO-PLATFORM-DISTANCE to fix MAXIMUM-TRACTION-RATIO does not affect MACHINE-GROOVE-PRESSURE or its maximum. Adding hoist cables to fix MAXIMUM-MACHINE-GROOVE-PRESSURE tends to relieve a problem with MAXIMUM-TRACTION-RATIO, although the effect is not substantial enough to warrant its inclusion as a fix for this constraint.

As long as only one of the two constraints is violated, the local search for a solution based on isolated constraint violations is satisfactory. However, if both constraints are violated, the system might thrash. We added to the VT shell the ability to treat this latter situation as a special case and investigate fixes for the two in tandem. To do this investigation, VT required one additional piece of information. If both constraints cannot be remedied at the same time, our domain expert relaxes MAXIMUM-MACHINE-GROOVE-PRESSURE before violating MAXIMUM-TRACTION-RATIO. If both cannot be fixed,

IF there has been a violation of MAXIMUM-TRACTION-RATIO, THEN try a DECREASE BY-STEP of 1 inch of CWT-TO-PLATFORM-DISTANCE which has a preference rating of 1 because it CAUSES NO PROBLEM.

Try an UPGRADE of COMP-CABLE-UNIT-WEIGHT which has a preference rating of 4 because it CHANGES MINOR EQUIPMENT SIZING.

Try an INCREASE BY-STEP of 100 lbs. of CAR-SUPPLEMENT-WEIGHT which has a preference rating of 4 because it CHANGES MINOR EQUIPMENT SIZING.

Try an UPGRADE for MACHINE-GROOVE-MODEL which has a preference rating of 11 because it INCREASES MAINTENANCE COSTS.

IF there has been a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE,

THEN try a DOWNGRADE for MACHINE-GROOVE-MODEL which has a preference rating of 1 because it CAUSES NO PROBLEM.

Try an INCREASE BY-STEP of 1 of HOIST-CABLE-QUANTITY which has a preference rating of 4 because it CHANGES MINOR EQUIPMENT SIZING.

Try a DOWNGRADE of COMP-CABLE-UNIT-WEIGHT which has a preference rating of 4 because it CHANGES MINOR EQUIPMENT SIZING.

Try a DECREASE BY-STEP of 10 lbs. of CAR-SUPPLEMENT-WEIGHT which has a preference rating of 4 because it CHANGES MINOR EQUIPMENT SIZING.

Figure 15. Potential Fixes for Two Conflicting Constraints.

VT tries to minimize the violation of MAXIMUM-MACHINE-GROOVE-PRESSURE without violating MAXIMUM-TRACTION-RATIO.

Whenever a demon detects a violation of one of these constraints, VT checks to see if the other has been violated. If it has, it resets the values of all potential fix values to the last value they had before the first violation of either constraint. It then tries out potential fixes, making sure that it does not repeat a combination of them, in the following order of fix function: (1) helps both, (2) helps one and doesn't hurt the other, and (3) helps one but does hurt the other. In the third case, the system applies the fix in the direction intended to remedy the constraint most important to fix. If there is asymmetry in the amount of change in a bidirectional fix, as there was for CAR-SUPPLEMENT-WEIGHT discussed earlier, after fixing the most desired constraint, VT changes the value in the other direction by the largest amount that still leaves the first constraint unviolated.

Nowhere in the VT knowledge base did we observe a problem that might cause the declaration of a premature dead end. In most cases, a failure report cannot be premature because the fixes that cause downstream violations are the only possible fix at their point of origin. Thus, any dead end observed at the aggravated downstream point is unavoidable. This situation is true for hoist cable quantity and diameter. For the other cases, the aggravating fix is the most expensive alternative for its constraint violation and won't be implemented unless nothing else works at this point. Again, this situation means that any dead end downstream would be unavoidable.

If we had identified a chain of interacting fixes that might result in premature dead end, it would have been relatively simple to provide a customized treatment for the potential site of the dead end. The VT shell could be modified so that whenever a dead end were found for such a constraint violation, VT would go back and try more expensive fixes at the

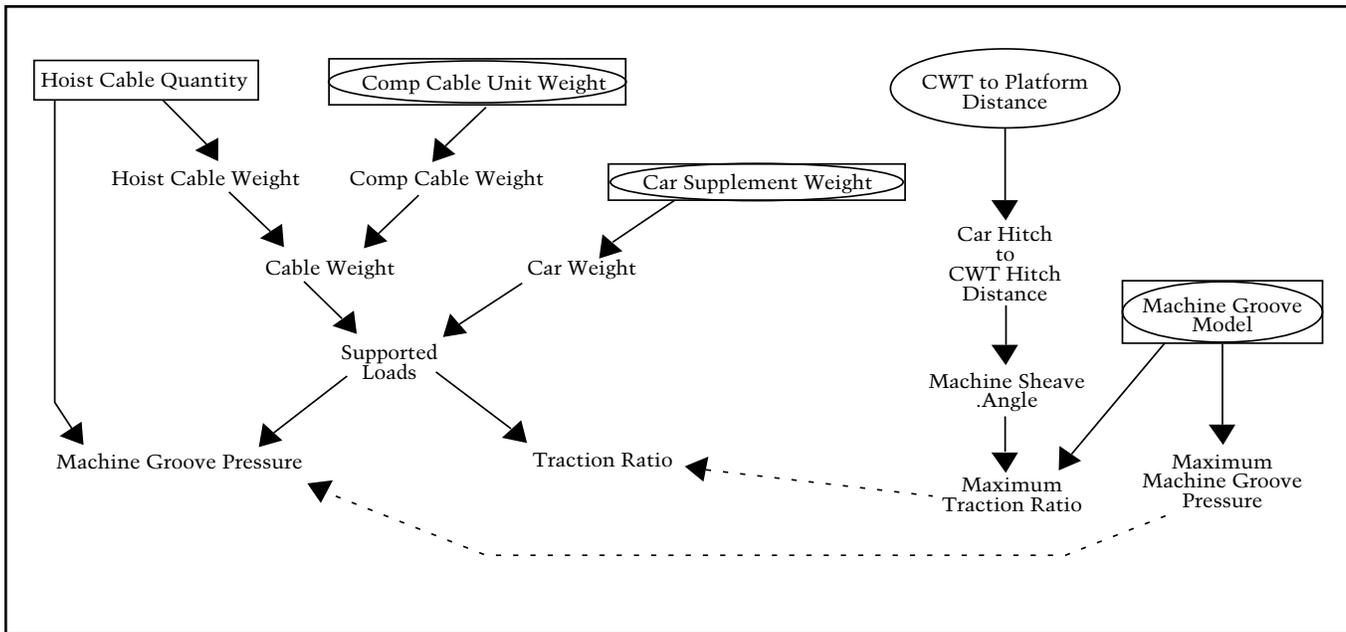


Figure 16. A Segment of the VT Knowledge Base Containing Antagonistic Constraints.

relevant prior constraint violation(s). SALT's map of interacting fixes could be used to identify the relevant prior fixes.

For VT then, SALT's analysis located cases in which fixes for different constraints interacted. Our examination showed in most cases the propagation of changes was such that a search based on fixing one constraint at a time would either converge on a solution or correctly announce no solution was possible. In three cases involving pairs of constraints, the system might thrash if constraint violations were fixed independently; so, additional knowledge was used to deal with the interacting constraint violations in combination.

Domain knowledge is needed to specify what revisions are possible in the real world and what their relative desirability is for fixing particular constraints. As a first step, SALT asks the domain expert to address each constraint violation individually. This situation relieves the expert from having to anticipate the ramifications for the rest of the design—something that is difficult for a person to do in a complex domain. SALT can help decide whether this approach is adequate for a problem solver because it has access to the entire knowledge base and because its representation of the knowledge base makes clear how the

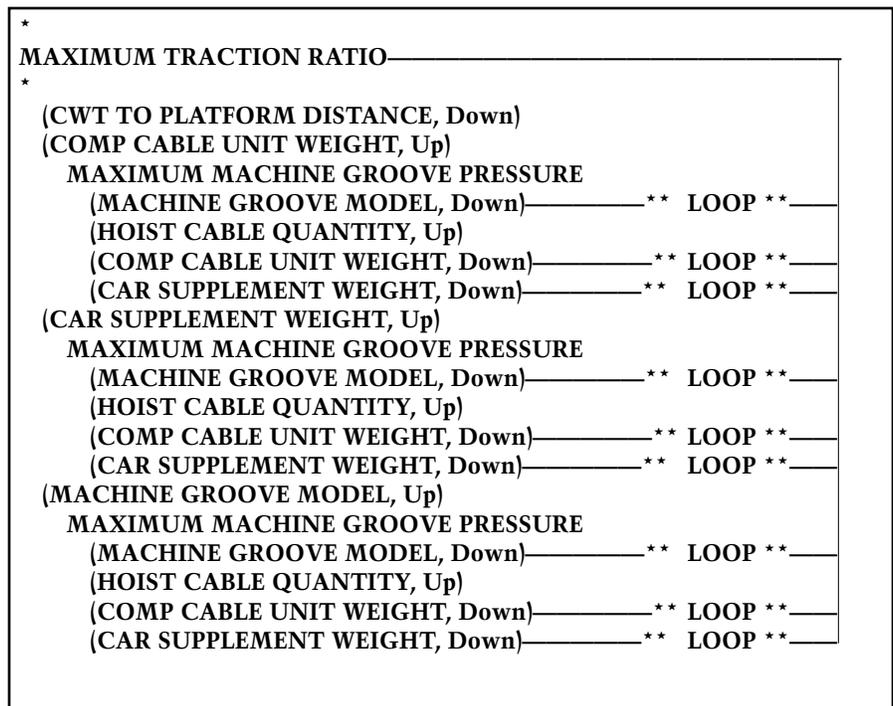


Figure 17. SALT's Report of Interacting Fixes.

knowledge is to be used. In the case of VT, a search space with hidden mine fields for a locally based search was much more manageable when supplemented with analysis-based special case treatment. The particular solutions to knowledge base inadequacies used in VT might not be sufficient for all constraint-satisfaction tasks. However, SALT's representation scheme

and analyses still help in addressing inadequacies because they make obvious the ramifications of problem-solving decisions with a given knowledge base. Thus, they can identify the need for additional knowledge and identify considerations that should go into deciding how and when knowledge should be used (Marcus 1988; Stout et al. 1987).

Comparison to Other Constructive Systems

The ordering of decisions in VT is in the spirit of the Expert Executive for aerospace vehicle design described in Chalfan (1986). The Expert Executive knows the inputs required and outputs produced by each of the procedures, or programs, it must configure. A program is run only when all other programs have been run whose outputs serve as its inputs. Unlike VT, the Expert Executive and the programs it configures are intended to be a design aid rather than a design expert. The Expert Executive and program configurations leave to the human expert the task of suggesting plausible starting values for free parameters, checking constraints, and directing revisions. VT performs these functions as well.

VT's architecture is probably most similar to that of EL, an expert system which performs analysis of electric circuits. EL makes a guess for, say, the current at a particular node and uses principles such as Ohm's Law and Kirchoff's Law to propose values at other points in the circuit. It is similar to VT in that it builds up a dependency network representing this propagation, backtracks whenever constraints are violated (when some point is assigned two different values), and uses a truth maintenance system. The main difference between EL and VT is that EL uses a domain-independent strategy of dependency-directed backtracking as opposed to VT's domain-specific knowledge-based approach. EL's decision of where to backtrack to is based solely on the dependency network's record of what guesses contributed to the conflicting constraints. Furthermore, EL is committed to a search that tries all possible combinations of all guesses, although it prevents thrashing by keeping track of combinations already tried and never repeating a combination. The related CONSTRAINTS language allows the user to direct backtracking and is similar to VT when running in interactive mode or performing what-if explanation.

Domain-independent dependency-directed backtracking is not satisfactory for VT's domain. VT is not sim-

ply searching for a single solution that meets constraints, where any solution is equally good. Generally, many possible solutions exist, and these solutions differ in domain-specific disadvantages. These differences are expressed in VT by using the expert's most preferred procedure to determine an initial value and using explicit preferences supplied by the expert on potential fixes for constraint violations.

GARI (Descotte and Latombe 1985) does incorporate a notion of domain-specific preference in its plausible reasoning but in an indirect and difficult-to-maintain manner. GARI's task is to devise a plan for machining parts that meets constraints on the order in which operations should be performed and the orientation of parts with respect to the machining tools. It employs backtracking whenever constraints conflict, and the decision about what point to backtrack to is determined by weights taken from domain experts. GARI backtracks to its most recent, lowest-weight decision. GARI does not use a dependency network or any relation of contribution in this decision. The result is that the decision it changes might be irrelevant to the constraint conflict which has arisen. In addition, although the weights are taken from domain experts, the designers note that the experts find the weights difficult to assign and that afterwards knowledge engineers must adjust these weights by experimentation. This process must be particularly difficult because these weights might have evolved to express both a combination of expense in terms of material, equipment cost, and so on, and of their likelihood to converge on a solution.

Two other design systems, AIR/CYL (Brown 1985) and PRIDE (Mittal and Araya 1986), use a knowledge-based approach to revising designs in response to constraint violations but differ somewhat from VT in the knowledge used. AIR/CYL has failure handlers that respond to constraint violations by calling for redesign of particular parts, or values, of the design. If more than one value might be revised, AIR/CYL uses a least backup strategy; it attempts revi-

sion at the most recently established relevant value. AIR/CYL moves back to the next most recently established only if it fails to remedy the violation at the current point, and so on. Brown wants to restrict the range of backtracking on the grounds that this restriction is what human design experts do. PRIDE also uses domain expertise to suggest how to revise parts of the design in response to constraint violations. For PRIDE, the presence of more than one suggestion about how to respond to a particular constraint violation causes the system to set up multiple contexts for exploring each suggestion. The PRIDE user can then select among alternatives. VT explores design revisions sequentially. In interactive mode, users can determine the order in which revisions are explored and suggest revisions of their own. In the absence of user input, VT has domain expertise regarding the preference of alternative fixes that it uses to decide the order in which it explores them.

R1 (McDermott 1982) is a system that constructs a solution but uses a strategy for plausible reasoning which might be described as "lookaround." Whenever a decision based on partial information is required, R1 tries to collect as much information as it can to ensure that the decision is acceptable. The kind of information it collects might be the same kind of information that could be used to augment fix knowledge, that is, information about how close the current solution is to violating related constraints. Without the kind of dependency network representation that VT/SALT uses, it is difficult to identify the role of this information. R1 is currently being revised to more clearly represent the roles that knowledge plays with respect to its own problem-solving method (van de Brug, Bachant, and McDermott 1986). This revision should make it easier to compare the two systems.

As mentioned at the outset, VT does postpone decisions where possible, but most of its effort goes into plausible guessing combined with backtracking. This system contrasts with MOLGEN whose main effort is put into managing its least commitment planning. Although MOLGEN

has the ability to backtrack, its guessing and backtracking capability is underdeveloped, and MOLGEN often does not recover from bad guesses (Stefik 1981a).

ISIS (Fox 1983; Smith, Fox, and Ow 1986) is another constraint-satisfaction planner that uses least commitment in job shop scheduling. ISIS expresses preferences as constraints. When forced to guess, that is, to choose among constraints it will meet when it can't meet all of them, ISIS conducts a beam search by maintaining in parallel the most preferred solutions. If a solution is not found by scheduling in the forward direction, that is, from first operation in time to last, then a second attempt is made starting from the last operation. The efficiency and probability of the search's success depends on the weights assigned to the constraints and the width of the beam. As with GARI, this architecture can lead to a difficult problem in credit assignment.

MOLGEN, ISIS, AIR/CYL, and PRIDE share the property of being hierarchical in that they select a metalevel plan or design and then refine it. In Friedland's version of MOLGEN especially, selecting which metalevel plan to refine involves a great deal of search (Cohen and Feigenbaum 1982). Although solution paths for extending a design for an elevator can differ depending on input parameters, these path differences are represented in VT as preconditions on individual steps. Nowhere are the path differences represented as separate metalevel designs. In the hierarchical planners, an abstract, metalevel design also serves to split the task into nearly independent subproblems. Interactions take the form of constraints that propagate from one subproblem to others. VT does not have a subtask level of organization to group procedures for extending a design and specifying constraints. One benefit of a subdivided architecture might be that it helps the system builders keep track of interactions among decisions. SALT's knowledge representation and the analysis it does based on the anticipated problem-solving strategy serves this function for VT (see also Marcus 1988).

VT's Performance

VT is currently in use at Westinghouse Elevator Company. It must function with a large knowledge base and converge on an acceptable solution within a reasonable amount of time. This section provides a description of its size and some indication of its performance characteristics.

Rule Characteristics

Because VT is implemented in OPS5 (Forgy 1981), it is appropriate to describe its size and complexity in terms of rules. VT currently has 3123 total rules. Of these, 2191 are domain-specific rules generated by SALT (70.2 percent). The remainder belong to the general shell for I-O, explanation, and problem-solving control. There are several types of SALT-generated rules. Some are not directly used in problem solving. These 698 rules (31.9 percent of all SALT-generated rules) contain domain-specific information required for I-O and the explanation facility. The remaining 1393 SALT-generated rules break down into the following categories: (1) 521 (23.8 percent) are forward-chaining rules for proposing a part of the elevator design, (2) 120 (5.5 percent) are forward-chaining rules for specifying constraints on the design, (3) 58 (2.7 percent) are rules for proposing potential fixes conditioned on the violation of particular constraints, (4) 44 (2.0 percent) are rules for directing exploration of the implications of a fix (lookahead), (5) 530 (24.2 percent) are lookahead rules for extending a design, and (6) 120 (5.5 percent) are lookahead rules for specifying constraints.

These rules represent procedures derived from the knowledge SALT collects in its three knowledge roles. The first three rule types make use of the knowledge in the roles of PROPOSE-A-DESIGN-EXTENSION, IDENTIFY-A-CONSTRAINT, and PROPOSE-A-FIX, respectively. The next group, rules for directing lookahead, define which procedures for proposing design extensions and identifying constraints are relevant to deciding whether proposed fixes actually remedy the constraint violation they are intended to fix. The last two categories employ the same knowledge encoded in the

first two groups, PROPOSE-A-DESIGN-EXTENSION and IDENTIFY-A-CONSTRAINT. They differ from the first two in that the conditions under which they fire are set up by the rules that direct lookahead. They are used to selectively explore implications of proposed fixes before choosing one to implement. Table 1 gives an impression of rule complexity in each of these categories.

Run Characteristics

Statistics reported here are based on a sample of six test cases that Westinghouse engineers feel are representative of the range of complexity which VT must handle. A breakdown of these cases on measures that reflect search complexity is given in table 2. All constraint violations are fixed on the runs in table 2; that is, there are no dead ends.

The breakdown of rule firings shown in table 3 helps to give an idea of where the activity is focused during a run. The breakdown for these jobs in CPU time, as measured on a VAX 11/780 with 20MB of memory, is shown in table 4.

Conclusion

VT is an expert system whose domain requires plausible guessing. Its problem-solving strategy incrementally constructs an approximate elevator design by proposing values for design parameters. At the same time, it identifies constraints on design parameters. If a constraint is violated, VT uses domain expertise to figure out how to revise the proposed design. In doing so, it uses an architecture that makes clear the role which each piece of domain-specific knowledge plays in proposing, constraining, and revising solutions. This knowledge representation serves as the basis for VT's explanation facility that can both explain past decisions and hypothesize about alternative solutions. It is also the foundation of an automated knowledge-acquisition tool, SALT, that can be used to generate expert systems which use this problem-solving strategy and explanation facility. SALT was used to acquire the knowledge for and to generate the system described here as well as to map out potential inter-

actions among fixes. This analysis helps a developer assess the potential for the system to converge on a solution if one exists. Trouble spots located by this analysis can be given special treatment in the backtracking search. In the future we plan to continue our exploration of the use of knowledge-based backtracking through the use of SALT as a tool to acquire the knowledge for other types of constructive tasks.

Acknowledgments

This research was sponsored by Westinghouse Elevator Company, Randolph, New Jersey. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Westinghouse Elevator Company. Many people have helped with VT's development. We would especially like to thank John Gabrick, Michael Gillinov, Robert Roche, Timothy Thompson, Tianran Wang, and George Wood.

References

Brown, D. 1985. Failure Handling in a Design Expert System. *Computer-Aided Design* 17:436-442.

Chalfan, K. M. 1986. A Knowledge System That Integrates Heterogeneous Software for a Design Application. *AI Magazine* 7(2): 80-84.

Cohen, P. R., and Feigenbaum, E. A. 1982. *The Handbook of Artificial Intelligence, Volume 3*. Los Altos, Calif.: Kaufmann.

Descotte, Y., and Latombe, J. C. 1985. Making Compromises among Antagonist Constraints in a Planner. *Artificial Intelligence* 27:183-217.

Doyle, J. 1979. A Truth Maintenance System. *Artificial Intelligence* 12:231-272.

Forgy, C. 1981. OPS5 User's Manual, Technical Report, CMU-CS-81-135, Dept. of Computer Science, Carnegie-Mellon Univ.

Fox, M. S. 1983. Constraint-Directed Search: A Case Study of Job-Shop Scheduling, Technical Report, CMU-CS-83-161, Dept. of Computer Science, Carnegie-Mellon Univ.

Marcus, S. 1988. A Knowledge Representation Scheme for Acquiring Design Knowledge. In *Artificial Intelligence Approaches to Engineering Design*, eds. C. Tong and D. Sriram, Forthcoming. Reading, Mass.: Addison-Wesley.

Marcus, S., and McDermott, J. 1986. SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems, Technical Report, CMU-CS-86-170, Dept. of Com-

Rule Type	Condition Elements	Attributes per CE	Action Elements
Extend a design	3.74	2.06	3.48
Identify a constraint	3.42	2.03	3.74
Propose a fix	2.24	3.31	1.07
Direct to lookahead	1.00	1.00	5.36
Extend an exploratory design	5.31	1.99	3.23
Identify an exploratory constraint	5.39	1.94	3.29

Table 1. Rule Complexity.

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Distinct constraints violated	7	8	8	12	9	12
Total constraint violations	9	9	12	16	17	23
Fixes explored per constraint violation	1.0	1.3	1.0	1.4	1.2	1.3
Nonconstraint values undone per implemented fix	18.9	25.7	26.0	33.7	29.6	40.4
Constraints undone per implemented fix	3.4	3.9	4.2	12.6	11.2	11.0

Table 2. Complexity Measures on Test Case Runs.

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
SALT-Generated Rules						
Extend a design	821	868	1050	1777	1358	2545
Identify a constraint	239	227	268	360	405	627
Propose a fix	9	9	12	16	17	23
Direct to lookahead	9	11	9	28	27	25
Extend an exploratory design	57	93	52	378	237	356
Identify an exploratory constraint	5	9	5	6	19	18
Subtotal	1140	1217	1396	2565	2063	3594
General Control Rules						
Test a constraint	147	147	189	232	250	422
Control a fix	472	592	594	1393	1251	1664
Maintain consistency	831	1074	1422	2886	2570	4732
Other	222	372	308	806	726	862
Subtotal	1672	2185	2513	5317	4797	7680
Total	2812	3402	3909	7882	6860	11274

Table 3. Rule Firings per Run.

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Time in forward-chaining mode	4:52	4:17	6:23	7:10	7:19	10:40
Time in fix-exploration mode	2:16	2:53	3:32	8:39	7:26	11:09
Total time per run	7:08	7:10	9:55	15:49	14:45	21:49

Table 4. CPU Time per Run.



Halbrecht Associates, Inc.

Executive Search Consultants

SERVING THE ARTIFICIAL INTELLIGENCE COMMUNITY

Halbrecht Associates, founded in 1957, was the first search firm in the United States to specialize in Information Systems Management, Operations Research, Management Sciences, Telecommunications, Computer Sciences and Decision Support Systems. We are now also in the forefront of the Artificial Intelligence field. Because we have been so extensively involved in this emerging technology, our portfolio of people includes extensive numbers of AI professionals at all levels, from the United States and abroad.

Our client list includes companies coast to coast, large and small, including start-ups with equity opportunities. Accordingly, depending on your needs, we are in a unique position to assist you rapidly with any of your requirements.

Our current assignments range downward from several Chief Scientists, or those who can provide the technical leadership to head up major AI programs, to Senior LISP Programmer. If you would like to be considered to be part of our AI data base, please send us your credentials. This will enable us to give you first notice of appropriate opportunities. All inquiries are in strictest confidence.

Contact: Daryl Furno, Senior Associate

1200 Summer Street • Stamford, CT 06905 • 203-327-5630

For free information, circle no. 193

puter Science, Carnegie-Mellon Univ.
puter Science, Carnegie-Mellon Univ.

Marcus, S.; McDermott, J.; and Wang, T. 1985. Knowledge Acquisition for Constructive Systems. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 637-639. Los Altos, Calif.: Morgan Kauffmann.

McDermott, J. 1982. R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence* 19: 39-88.

Mittal, S., and Araya, A. 1986. A Knowledge-Based Framework for Design. In Proceedings of the Fifth National Conference on Artificial Intelligence, 856-865. Los Altos, Calif.: Morgan Kauffmann.

Smith, S.; Fox, M.; and Ow, P. 1986. Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems. *AI Magazine* 7(4): 45-60.

Stallman, R. M., and Sussman, G. J. 1977. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artifi-*

cial Intelligence 9: 135-196
cial Intelligence 9: 135-196.

Stefik, M. 1981a. Planning and Meta-Planning (MOLGEN: Part 2). *Artificial Intelligence* 16: 141-170.

Stefik, M. 1981b. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence* 16: 111-140.

Stefik, M.; Aikins, J.; Balzer, R.; Benoit, J.; Birnbaum, L.; Hayes-Roth, F.; and Sacerdoti, E. 1983. The Architecture of Expert Systems. In *Building Expert Systems*, eds. F. Hayes-Roth, D. Waterman, and D. Lenat, 89-126. Reading, Mass.: Addison-Wesley.

Stout, J.; Caplain, G.; Marcus, S.; and McDermott, J. 1987. Toward Automating Recognition of Differing Problem-Solving Demands. Paper presented at AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems.

Sussman, G. J. 1977. Electrical Design, A Problem for Artificial Intelligence Research. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 894-900. Los Altos, Calif.:

Morgan Kauffmann.
Morgan Kauffmann.

Sussman, G. J., and Steele, G. L., Jr. 1980. CONSTRAINTS—A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence* 14: 1-39.

van de Brug, A.; Bachant, J.; and McDermott, J. 1986. The Taming of R1. *IEEE Expert* 1: 33-38.

Notes

1. The down in downgrade usually pertains to a decrease in size or cost. In the VT domain, size tends to vary inversely with preference.

2. A related but less serious problem is that a remedy not chosen might have an ameliorating effect on a downstream constraint violation. In such a case, the system might miss a solution in which the total cost of fixing the two violations might be less if a more costly fix were chosen for the first.