Integrated Planning for Automated Image Processing

Steve Chien, Helen Mortensen, Christine Ying, and Shouyi Hsiao
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive M/S 525-3660
Pasadena, CA 91109-8099
(818) 306-6144 FAX (818) 306-6912
chien@aig.jpl.nasa.gov

Abstract

The Multimission VICAR Planner (MVP) system is an AI planning system which constructs executable image processing programs to support Operational Science Analysis (OSA) requests made to the Jet Propulsion Laboratory (JPL) Multimission Image Processing Subsystem (MIPS). MVP accepts as input: image files and a high-level specification of desired corrections, enhancements, output properties (such as for mosaics). MVP then derives: unspecified but required processing steps, relevant image processing library programs, and appropriate parameter settings for such programs - constructing an executable image processing program to fill the image processing request. MVP is currently available to analysts to fill requests and reduces the effort to fill radiometric correction, color triplet reconstruction, and mosaicking tasks by over an order of magnitude.

Problem Description

In recent times, improvements in spacecraft imaging hardware have caused a massive increase in the amount of scientific data and variety of science data products. Simultaneously, increased sophistication of image processing algorithms has complicated the image processing task. While ensuring physical access to the vast amounts of space-related data can be achieved, it is often extremely difficult for the average user to manageably prepare and process the available scientific data.

One method for reducing this data access, preparation, and analysis problem is the development of general purpose data processing languages and interfaces. These languages and interfaces allow users to access and process data within a common environment. For image processing, the VICAR environment (Video Image Communication and Retrieval [1]) (LaVoie et. al. 89) is a major constituent of JPL's image processing capability. VICAR provides a standard interface to allow a user to retrieve video image data and apply sophisticated image processing algorithms. The principal focus of the VICAR system is planetary imaging, and as such it supports imaging for JPL flight projects including VOYAGER, VIKING, MAGELLAN, GALILEO, CASSINI, etc. VICAR has been applied to other space imaging missions such as IBIS and LANDSAT. The VICAR system has also been applied to numerous other applications including: astronomy, earth resources, land use, biomedicine, and forensics. VICAR is a principal component of the Multimission Image Processing Laboratory (MIPL). Outside of JPL, VICAR users include universities, the military, research institutions, aerospace corporations, companies, and Galileo HIIPS (home institution image processing subsystem) sites with a total user group of over 100 users.

VICAR allows individual image processing steps (called VICAR programs) to be combined into more complex image processing scripts called procedure definition files (PDFs). As one of their primary duties, JPL analysts construct PDFs to perform image correction, image enhancement, construct mosaics, and to create movies and render objects. Individual processing programs perform functions such as:

1. photometric correction - correcting the image for lighting conditions due to the position of the sun relative to the imaging device and target,
2. radiometric correction - correcting for varying camera response depending on camera state and other properties such as where in the field of view the image is read,
3. line fillin - interpolating missing lines caused by data transmission errors.

In order to fulfill OSA requests for image processing, analysts must create complex VICAR programs, determining relevant programs, order of execution, and parameter settings using their knowledge of the processing steps and processing program requirements.

Unfortunately, manual construction of VICAR programs is both labor and knowledge intensive.

[0]This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Other past and present members of the MVP team are Alex Gray, Joe Nieten, and Jean Lorre.

[1]This name is somewhat misleading as VICAR is used to process considerable non-video image data such as MAGELLAN synthetic aperture radar data.

```
LOCAL (camera8,filter8,exprng8) STRING
LOCAL (sc6,scan_rate8,fds8) INTEGER
CAMPARAM /proj/mvp/images/1972833.spk SC=sc6 SCAN=scan_rate8 CAMERA=camera8 FILTER=filter8 FDS=fds8
EXPRNG=exprng8
!! compute voyager radiometric callibration file
LET cal2 = "VGR00" // "&sc6" // "A" // ":[" // "&camera8" // "]" // "PDF." // "&filter8"
LOCAL (simult_image_mode7,planet9) STRING
let simult_image_mode7 =  !! fill in a non-null string if file /proj/mvp/images/1972833.spk is simultaneous imaging mode
let planet9 =  !! fill in planet for image file /proj/mvp/images/1972833.spk
          !! one of ("jupiter" "saturn" "uranus" "neptune")
!! compute voyager dark callibration file
LOCAL dark2, scan_string7 STRING
if (jupiter = "JUPITER")
  let jupiter = "J"
else-if (jupiter = "SATURN")
  let jupiter = "S"
else-if (jupiter = "URANUS")
  let jupiter = "U"
else-if (jupiter = "NEPTUNE")
  let jupiter = "N"
end_if
if (2.0 = 10)
  let scan_string7 = "A"
else
  let scan_string7 = "&2.0"
end_if
if (simult_image_mode7 <> "")
  let simult_image_mode7 = "S"
end-if
let dark2 = "VGR00" // "&sc6" // "A" // ":[" /A "&camera8" /A "]" // "I" // "&fds8" // "." // "&jupiter" // "&scan_string" //
"&simult_image_mode7"
FICOR77 INP=(/proj/mvp/images/1972833.spk, &cal2, &dark2) OUT=&/proj/mvp/images/1972833.fic GAIN=0
```

Figure 1: Sample VICAR Code Fragment

The VICAR procedure generation problem is also a knowledge intensive task in that an analyst must possess knowledge of:

1. image processing and image processing programs (as of 1/93 there were approximately 50 frequently used programs, some having scores of options)
2. database organization and database label information to understand the state of relevant data
3. the VICAR programming language to produce and store relevant information.

For example, shown below is the MVP-generated code fragment to radiometrically correct a VOYAGER image file. In this case, the program to radiometrically correct a VOYAGER image file reuires a radiometric callibration file, which can be determined if one knows the spacecraft (VOYAGER1 or VOYAGER2), camera(narrow angle or wide angle), and filter relevant to the image. Determining the dark current callibration file requires knowing the spacecraft, camera, fds (time) of the image, planet target of the image, the scan rate, and if the camera was in sumltaneous imaging mode. MVP represents these requirements and uses the requirements and knowledge of how to extract imaging parameters from the image file (as much as possible) to generate the code fragment as listed.

Because of the complexity and amount of program knowledge relevant to the task as well as the many interacting problem goals, VICAR procedure generation is a labor intensive task. Generation of a complex VICAR procedure may take up to months of analyst time.

One difficulty facing analysts is the diversity of knowledge required to produce expert VICAR procedures. While certain VICAR users, such as expert analysts, may possess much of this knowledge, the vast majority of VICAR users are novice to one or more aspects of this knowledge. Unfortunately, this increases the difficulty of data access and preparation and increases the load on experts who must spend a significant amount of their time assisting those less knowledgable. For example, a university user may know a great deal about the science behind the imaging and the theory behind the processing steps but may know little about the underlying assumptions of the implementation of the processing steps or VICAR itself. Similarly, a programmer who writes processing programs may know quite a bit about their particular program but may experience difficulty in writing a VICAR procedure to generate data to test his or her program. This great need for VICAR knowledge exists because of the significant time it takes to become proficient in multiple aspects of VICAR. Generally, a VICAR user with 1-2 years of experience is considered a novice VICAR user, while it may take 4-5 years to become a VICAR expert.

Application Description

MVP (Chien94a; Chien94c) partially automates generation of image processing procedures from user requests and a knowledge-based model of an image processing area using Artificial Intelligence (AI) automated planning techniques (Iwasaki & Friedland 85; Pemberthy & Weld 92; Stefik 81). In AI planning, a system uses: 1) a model of actions in a domain; 2) a model of the current state; and 3) a specification of the desired state; to reason about what actions to take to achieve some specified goals. In VICAR image processing the actions are VICAR image processing programs, the current state is the current state of the image files of interest, and the specification of the desired state corresponds to the user image processing goals. By partially automating the filling of basic science image processing requests, image processing request turnaround time will be reduced, analysts time will be freed for more complex and challenging science requests, and analyst workload will be reduced.

The MVP Architecture

The overall architecture for the MVP system is shown in Figure 2. The user inputs a problem specification consisting of processing goals and certain image information using a menu-based graphical user interface. These goals and problem context are then passed to the decomposition-based planner. The decomposition-based planner uses image processing knowledge to classify the overall problem type which the user has specified in a process called *skeletal planning* (Iwasaki & Friedland 85). This classification is then used to decompose the problem into smaller subproblems in a process called *hierarchical planning* (Stefik 81). The subproblems produced by the decomposition process are then solved in a process called *operator-based planning* (Pemberthy & Weld 92), in which a planner uses a description of possible actions (in this case image processing steps) to determine how to achieve subproblem goals as indicated by the problem decomposition. The resulting plan segments are then assembled using constraints derived in the decomposition process. The resulting plan is then used to generate an actual executable VICAR PDF using conventional code-generation techniques (such as

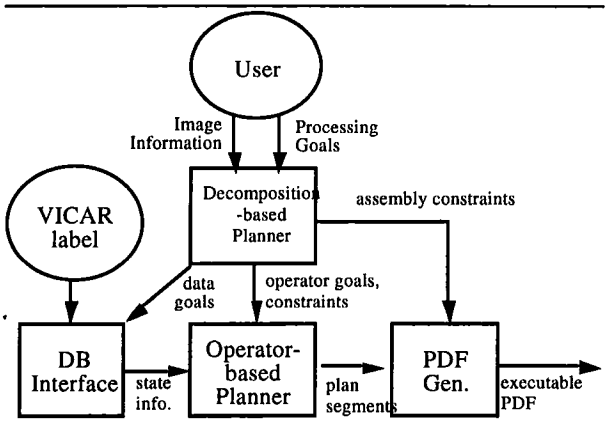

Figure 2: MVP System Architecture

macro-expansion).

MVP uses both decomposition and operator-based planning paradigms for two reasons: search control and user understandability. Plans in the MVP domain can be of considerable length (up to 100 steps) and each step (or VICAR program) can involve reasoning about numerous complex effects (many operators have tens of effects). Due to the large search space caused by this complexity, conventional operator-based planning approaches are not able to tractably construct plans in the VICAR domain without significant control knowledge. By using the decomposition planning paradigm, MVP breaks up the large search space planning problems caused by the complexity of the image processing problems in to multiple smaller problems, thus reducing the search problems encountered during operator-based planning. Indeed, the problem decomposition rules used in MVP can be considered a very important form of search control knowledge essential to MVPs image processing capability.

MVP also uses decomposition-based planning for reasons of user understandability. Even if a purely operator-based planning approach were able to generate plans to solve the VICAR problems, these plans would be difficult for MIPL analysts to understand because MIPL analysts do not consider an entire image processing problem all at once. Typically, analysts begin by classifying the general problem being addressed into one of a general class of problems, such as mosaicking, color triple processing, etc. They then use this classification and the problem context to decompose the plan into several abstract steps, such as local correction, navigation, registration, touch-ups, etc. Because MVP uses decomposition-based planning to reduce the original image processing problem, it can easily produce an annotated trace of how the overall problem was classified and decomposed, simplifying analyst understanding of the plan generation process.

Skeletal and Hierarchical Planning Using Decompositions

MVP integrates decomposition and operator based planning paradigms. MVP first reduces a problem using decomposition methods, then solves the resulting subproblems using operator planning techniques. MVP uses knowledge represented as decomposition rules to perform skeletal and hierarchical planning.

Skeletal and Hierarchical Planning in MVP Skeletal planning (Iwasaki & Friedland 85) is an approach to planning which casts planning as a structured classification problem. In skeletal planning, a planner identifies a new problem as one of a general class of problems based upon the goals and initial state. This technique was originally developed as a model of experiment design in molecular biology; however, skeletal planning is also an accurate model of how expert analysts attack VICAR procedure generation problems. Typically, in a VICAR problem, there is a central goal for processing, such as mosaicking, which then dictates a decomposition of the overall problem into subproblems such as local correction, navigation, and registration. MVP attacks a VICAR problem by first determining the general problem class, and then using this problem class to perform an initial decomposition of the top-level image processing goals.

Hierarchical planning (Stefik 81) is an approach to planning where abstract goals or procedures are incrementally refined into more and more specific goals or procedures as dictated by goal or procedure decompositions. MVP uses this approach of hierarchical decomposition to refine the initial skeletal plan into a more specific plan specialized based on the specific current goals and situation. This allows the overall problem decomposition to be influenced by factors such as the presence or absence of certain image calibration files or the type of instrument and spacecraft used to record the image. For example, geometric correction uses a model of the target object to correct for variable distance from the instrument to the target. For VOYAGER images, geometric correction is performed as part of the local correction process, as geometric distortion is significant enough to require immediate correction before other image processing steps can be performed. However, for GALILEO images, geometric correction is postponed until the registration step, where it can be performed more efficiently.

Decomposition-based Planning in MVP MVP uses a decomposition approach (Lansky93; Erol et al. 94) to perform Skeletal and Hierarchical planning. In a decomposition approach, decomposition rules dictate how to break a problem into smaller problems. In many cases, it is possible to decompose a problem in several ways. In these cases, the planner then searches the space of possible decompositions. Decomposition approaches are extremely powerful in that many other paradigms (such as modal truth criterion planning (Lansky93) can be implemented in a decomposition-based approach. The syntax for a decomposition rule is shown in Figure 2.

LHS		RHS
G_I = initial goal set/actions		G_R = reduced goal set/actions
C_0 = constraints	==>	C_1 = constraints
C_2 = context		N = notes on decomposition

Figure 3: Decomposition Rule Syntax

LHS	RHS	
G_I= mosaicking goal present	G_R =	1. local correction, 2. navigation 3. registration 4. mosaicking 5. touch-ups
C_0= null		
C_2= an initial classification has not yet been made	C_1 =	these subtasks be performed in order 1. 2. 3. 4. 5.
	N =	protect local correction until mosaicking problem class is mosaicking

Figure 4: Skeletal Planning Decomposition Rule

This rule states that a set of goals or actions G_I can be reduced to a new set of goals or actions G_R if the set of constraints C_0 is satisfied in the current plan and the context C_2 is satisfied in the current plan provided the additional constraints C_1 are added to the plan. Skeletal planning in MVP is implemented in by encoding decomposition rules which allow for classification and initial decomposition of a set of goals corresponding to a VICAR problem class. The LHS of a skeletal decomposition rule in MVP corresponds to a set of conditions specifying a problem class, and the RHS specifies an initial problem decomposition for that problem class. For example, Figure 3 shows a decomposition rule for the problem class *mosaicking with absolute navigation.*

The simplified decomposition rule shown in Figure 3 states that if mosaicking is a goal of the problem and an initial problem decomposition has not yet been made, then the initial problem decomposition should be into the subproblems local correction, navigation, etc. and that these steps must be performed in a certain order. This decomposition also specifies that the local correction goals must be protected during the navigation and registration processes. In general, MVP permits goals and abstract steps to be specified in the G_I and G_R fields. The constraints C_0 and C_1 and context specify restrictions on when the rule is applicable, and include : 1. constraints on the ordering of steps or goals; 2. constraints on the assignment of variables representing objects in the plan; goals or steps that must be present in the plan; and 4. goals or steps that are not allowed to be present in the plan. Hierarchical planning is also implemented within the decomposition framework. In this case the LHS specifies a context in which a set of goals or actions can be decomposed into a lower level set of goals or actions. For example, the decomposition rule in Figure 4 states that if the limb is present in all of the images (meaning that the sun-facing edge of the planet is visible in all of the images), for VOYAGER and GALILEO images, the navigation step can be performed by absolute navigation (a process in which each of the images can be navigated independently).

LHS	RHS
G_I= navigation action present	G_R = 1. absolute navigation
C_0= null	
C_2= the project is VOYAGER or GALILEO and limb present in all images	C_1 = null
	N = null

Figure 5: Hierarchical Refinement Decomposition Rule

This decomposition-based approach to skeletal and hierarchical planning in MVP has several strengths. First, the decomposition rules very naturally represent the manner in which the analysts attack the procedure generation problem. Thus, it was a relatively straightforward process to get the analysts to articulate and accept classification and decomposition rules for the subareas which we have implemented thus far. Second, the notes from the decomposition rules used to decompose the problem can be used to annotate the resulting PDF to make the VICAR programs more understandable to the analysts. Third, relatively few problem decomposition rules are easily able to cover a wide range of problems and decompose them into much smaller subproblems.

Operator-based Planning in MVP

MVP uses classical operator-based planning techniques to solve subproblems produced by the decomposition-based planner. An operator-based planner uses:

1. a model of actions M (in this case the model represents the requirements and effects of individual VICAR steps);
2. a specification of a current state C (this corresponds to the current database state); and
3. a specification of a goal criteria G (this corresponds to user request specification)

to derive:

a sequence of actions A, that when executed in the current state C, result in a state which satisfies the goal criteria G. In this case A will correspond to the VICAR script the user can execute to perform the image processing task at hand.

In operator-based planning, an action is represented in terms of its preconditions (those things required to be true before an action can be executed), and its effects (those things true after an

action is executed). For example, in VICAR image processing, the program GALSOS is used to radiometrically correct Galileo image files. This would be represented by a planning action for the GALSOS program, which could be applied to an image file. This action would have the precondition that the image file be a Galileo image file. This action would also have the effect that the image file is radiometrically corrected after GALSOS has been run.

When constructing a plan to achieve a goal G1, a planner will consider those actions which have G1 as an effect. Thus, if the planner wanted to achieve that a particular image file was radiometrically corrected, it would consider applying the VICAR program GALSOS on the image file. If a planner decides to add an action A1 to a plan to achieve a goal, it will then have to achieve all of the preconditions of A1. This process is called *subgoaling*. For example, the VICAR program PTP requires that an image file be in byte format before PTP can be applied. Thus if the planner decides that it wants to apply the PTP program to a file, it then must ensure that the image file is in byte format. In some cases this will already be true, in other cases running a programs to change the file format may be required.

Planning is also complicated by the fact that there are typically interactions between subparts of the plan. Thus, actions introduced to achieve goals in one part of the plan may undo goals achieved in another part of the plan. The process of ensuring that such interactions do not occur is called *protection*. Protection can involve such measures as ensuring that the goal is no longer needed when it is undone, or ensuring that the offending action effect does not in fact refer to the same object as the achieved goal (by creating a copy of a file, for example). We have only briefly sketched some of the elements of operator-based planning, for a more detailed treatment of operator-based planning algorithms the reader is referred to (Pemberthy & Weld 92; Chapman87).

To illustrate the operator-based planning process, consider the (simplified) image processing operators shown in Figure 5. This information can be summarized by the information shown below indicating the relevant programs for achieving the goals of missing line fillin, spike removal, and radiometric correction for Voyager and Galileo images. When constructing a plan to achieve these goals, depending on the project of the image file (e.g., either Voyager or Galileo), MVP can determine the correct program to use because the preconditions enforce the correct program selection.

	Voyager	Galileo
fillin missing lines	VGRFILLIN	GLLFILLIN
remove spikes	ADESPIKE	ADESPIKE
radiometric corr.	FICOR77	GALSOS

However, determining the correct ordering of actions can sometimes be complex. In this case, the correct order to achieve the goals of line fillin, spike removal, and radiometric correction is dependent upon the project of the file. In the case of Voyager files, ADESPIKE (spike removal) requires raw pixel values and FICOR77 (radiometric) changes pixel values to correct for camera response function, so FICOR77 removes a necessary condition for ADESPIKE. This interaction can be avoided by enforcing that ADESPIKE occurs before FICOR77. VGRFILLIN requires binary EDR header on the image file which is not maintained by ADESPIKE, this interaction can be avoided by requiring VGRFILLIN to be executed before ADESPIKE.

The Galileo case is slightly different. GALSOS undoes missing line fillin so that it interferes with GLLFILLIN. This interaction can be avoided by enforcing GLLFILLIN after GALSOS. Additionally, GALSOS requires raw pixel values, and ADESPIKE alters the pixel values, so ADESPIKE interferes with this condition. This interaction can be avoided by requiring that GALSOS occur before ADESPIKE.

	Voyager	Galileo
Execution Order:	VGRFILLIN	GALSOS
	ADESPIKE	GLLFILLIN
	FICOR77	ADESPIKE

This simple example illustrates the types of interactions and context-sensitivity that the VICAR image processing application entails. All of these interactions and context sensitive requirements are derived and accounted for automatically by MVP using the operator specification, thus allowing construction of plans despite complex interactions and conditions.

MVP also uses operator-based planning techniques to determine correct program option settings. MVP uses preconditions to represent various program option settings and the situations under which they will achieve desired effects. Thus, when an action is selected to achieve a goal, the correct program option settings have also automatically been determined.

Knowledge Acquisition and Refinement in MVP

In order for MVP to be able automatically generate VICAR image processing procedures, the knowledge base for MVP must represent large amounts of knowledge in the form of decomposition rules and operators. Unfortunately, eliciting and encoding this knowledge is a tedious, time-consuming task. In order to facilitate this key process of knowledge acquisition and refinement we have been developing a set of knowledge-base editing and analysis tools. These tools can be categorized into two general types: (1) static knowledge base analysis tools; and (2) completion analysis tools. Because MVP uses two types of knowledge: decomposition rules and operator definitions, each of these tools can be used with each of these representations. We describe the capabilities of these tools below.

Operator	VGRFILLIN	GLLFILLIN	ADESPIKE	FICOR77	GALSOS
Preconditions	VGR image EDR present	GLL image	(GLL image) or ((VGR image) and (raw values))	VGR image	GLL image raw pixel values
Effects	missing lines filled in.....		spike removal	radiometric corr. blemish removal not raw values	radiometric corr. reed-solomon overflow corr. saturated pixel corr. not missing line fillin

Figure 6: Simplified Operator Definitions

Static Analysis Tools for Task Reduction Rules in MVP

Static analysis tools analyze the knowledge base to determine if pre-specified problem-classes are solvable. The static analysis techniques can be used in two ways: 1. fast run-time checking using propositional analysis; and 2. off-line knowledge-base analysis to verify that a domain theory can solve problems in the non-propositional case. In the propositional case, all actions and goals are considered only for the predicate or goal name, (e.g., "(radiometrically-corrected ?file1)" becomes "radiometrically-corrected") - thus simplifying the analysis considerably. The propositional analysis is used as a fast run-time checking component to catch simple errors when debugging a knowledge base. The full static rule analysis is extremely useful in verifying that a domain theory has appropriate coverage. For the MVP application, the set of allowable interface goals is formally specified, in terms of a logical expression on a set of goals produced by the interface (e.g., all combinations of these 5 goals, except that goal4 and goal3 are incompatible, and that every time goal 2 is selected goal 1 must have this parameter). The full static rule analysis is run on these allowable combinations and the decomposition rules Verified to approximately cover the combinations (this corresponds to exhaustive testing of the task reduction rules).

For both propositional and full static rule analysis cases, the knowledge engineer is required to define a problem context consisting of: 1. a set of input non-operational activities or goals that may be input to the system; and 2. and a set of operational activities or goals and the goal of the static rule analysis is to verify that the input goals/activities can be reduced into operational goals/activities.

StaticRuleAnalyze(input-goals, operational-goals, rules)

```
initialize plan queue
        Q = {(goals=input-goals, constraints={})}
select a plan P from Q
      for each plan P' produced by applying
      a task reduction rule
          - handling constraints as indicated below
          IF P' contains only operational goals/activities
          THEN return SUCCESS
          ELSE add P' to Q and continue
```

Constraint type	Propositional Case	Full Case
codesignation	ignored	tracked
not-present	ignored	tracked
present	propositional	tracked
protection	ignored	tracked

The principal difference between the propositional and non-propositional cases is that when predicates are transformed to the propositional case, constraint resolution optimistically presumes variable assignments will remove conflicts. For example, if one reduction rule has a not-present constraint (foo ?a) and there is an activity (foo ?c) in the plan, the planner would only allow the decomposition if ?a and ?c do not codesignate. However, in the propositional case, the reduction rule has a not-present foo constraint which is ignored - presuming that the parameters to foo can be chosen to remove the conflict. Likewise, if the reduction rule had a present constraint (foo ?a), the full case would be satisfied only if ?a and ?c codesignate, while the propositional case would presume the constraint was satisfied. In the propositional case protection constraints are ignored presuming that parameter selection will resolve the constraint (as in not-present constraints).

The graph below describes a problem specified to the static analysis for the relative navigation image processing subproblem. The italicized names at the top correspond to input non-operational goals. The non-italicized "GLL Image" corresponds to an operational input. The graph shows the decomposition of the input goals into the operational goals construct-om-auto/man-refined, temporal-perspective-correct, and display-om-error-auto/man-refined, which correspond to lower-level planning goals to be solved by the operator-based planner.

In this example, both the propositional and full static rule analyses would produce the same result - namely that the problem can be solved. Typically, the propositional analysis is set to run whenever rule sets are loaded, as it runs quickly enough to be transparent to the user. However, the full static rule analysis is typically run off-line on large sets of goal combinations to detect cases in which the reduction rules defined by the knowledge engineer are incomplete.

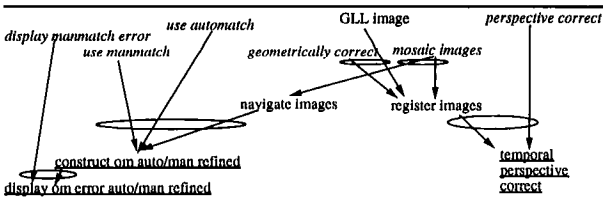

Figure 7: Static Rule Analysis Example

Static Analysis Tools for Operator-based Planning in MVP

The static analysis techniques can also be applied to the MVP's operator-based planner component. This is accomplished by generalizing the planning algorithm. In the propositional case, in order to treat the domain theory optimistically, we must assume that all protection interactions can be resolved by variable assignments, thus the propositional operator static analysis corresponds to the propositional rule-based static analysis where using an operator for an effect E corresponds to using a rule for the effect E with LHS of the operator preconditions and conditional preconditions of E. The non-propositional static analysis case is handled by modifying a standard operator-baed planner by adding an achievement operation corresponding to presuming any operational fact is true in the initial state. We are currently investigating using more sophisticated static analysis techniques to detect more subtle cases where goals are unachievable (Etzioni to appear; Ryu & Irani 92).

StaticOperatorAnalyzeNonPropositional
(input, operational, operators)
initialize Q = {(goals=input, constraints={})}
select a plan P from Q
    for each plan P' produced by achieving a goal
    using the following methods:
        1. using an existing operator in the plan
        2. adding a new operator to the plan
        3.* assuming an operational goal true
            in the initial state
    resolve conflicts in P' (protections)
    IF P' has no unresolved conflicts
       and has no unachieved goals
    THEN return SUCCESS
    ELSE add P' to Q and continue

Completion Analysis Tools in MVP

The second type of knowledge base development tool used in MVP is the completion analysis tool. In many cases, a knowledge engineer will construct a domain specification for a particular VICAR problem, test it out on known files and goal combinations. Two possible outcomes will occur. First, it is possible that the domain specification will produce an invalid solution, in this case the knowledge engineer can use the inconsistent part of the solution to indicate the flawed portion of the domain theory. For example, if a step S is missing, the user can focus on the final condition C or precondition P of another operator S was achieving. Alternatively, if the ordering of operators or variable assignments is not valid, the knowledge engineer can focus on the protection which should have been enforced. The second possibility is that the domain specification fails to allow the desired solution. In this case, detecting the flawed part of the knowledge base is more difficult, because it is difficult to determine which part of the domain specification caused the desired output plan to fail. Completion analysis tools directly address this problem. The completion analysis tools allow the decomposition or operator-based planner to construct a proof with assumptions that an extremely limited number of goals or subgoals can be presumed achievable (typically only one or two). By seeing which goals if assumable, make the problem solvable, the user can focus more quickly on the flawed portion of the knowledge base. In the operator-based planner, completion analysis is permitted by adding another goal achievement method which corresponds to assuming that the goal is magically achieved. When the planner exceeds resource bounds after finding a number of solutions, these solutions are then reported back to the user to assist in focussing on possible areas of the domain theory for refinement. The basic completion analysis tools for the task reduction planner and operator planner are shown below.

CompleteReductionPlanner (input, operational, rules)
initialize Q = (goals=input, constraints={})
select a plan P from Q
    for each plan P' produced by reducing P
    using a task reduction rule
        if the constraints in P' are consistent
          IF P' contains only operation goals/activities
          THEN return SUCCESS
          ELSE add P' to Q and continue
    for each plan P' produced by presuming
    the current goal achieved/operational
        IF P' has only operation goals/activities
        THEN return SUCCESS
        ELSE add P' to Q and continue;
              incrementing assumptions

CompleteOperatorPlanner(input, initial-state, operators)
initialize Q = (goals=input, constraints={})
select a plan P from Q
    for each plan P' produced by achieving a goal
    using the following methods:
        1. using an existing operator in the plan
        2. adding an operator to the plan
        3. use the initial state to achieve the goal
        4.* assume the goal true using completion
            analysis; increment assumptions
    resolve conflicts in P' (protections)
    IF P' has no unresolved conflicts
    and has no unachieved goals
    THEN return SUCCESS
    ELSE add P' to Q and continue

The main drawback of the completion analysis tools is that they dramatically increase the size of the

search space. Thus, with the completion analysis tools, the user can specify that only certain types of predicates can be presumed true, or predicates relating to certain operators. This has been fairly effective in focussing the search. Thus, the completion analysis techniques are generally used in the following manner. MVP automatically logs any problems unsolvable by the task reduction planner (unreducable) or operator-based planner (no plan found). The user then specifies that one of the top-level goals may be suspended. The completion planner then finds a plan which solves all but one of the top-level goals - focussing the user on the top-level goal which is unachievable. The user then determines which operator O1 that should be achieving the goal, and specifies that the completion planner may consider suspending preconditions of O1. After the planner then determines which precondition of O1 is preventing application of this operator, the user determines which operator should be achieving this precondition P1 of O1, and the process continues recursively until the flawed operator is found. For example, it may be that a protection cannot be enforced, thus preventing a precondition P from being achieved. Or it may be that no operator has an effect that can achieve P, or that the conditional effect that should be used has the wrong conditional preconditions. Th graph below illustrates this process from an actual debugging episode occurring in the development of the relative navigation portion of the planning knowledge base:

1. The user specified that a top-level goal may be suspended, and the completion planner constructed a plan achieving all goals but the top-level goal of (compute-om-matrix ?om-matrix ?file-list ?file-list).
2. The user then determined that the OMCOR2 operator should have been achieving this goal. The user then runs the planner allowing suspension of a precondition of the OMCOR2 operator.
3. The completion planner finds a plan achieving all goals except the OMCOR2 precondition (tiepoint-file ?tp ?file-list manmatch).
4. The user then determines that the precondition should be achieved by the manmatch operator, and runs the planner allowing suspension of one of the preconditions of manmatch.
5. The completion planner then finds a plan achieving all goals but the precondition (refined-overlap-pairs ?rop-file ?file-list) of manmatch.
6. The user then determines the precondition should have been achieved by the EDIBIS operator and runs the planner allowing suspension of an EDIBIS precondition.
7. The completion planner finds a plan achieving all goals but the precondition (crude-overlap-pair ?cop-file ?file-list) of EDIBIS.

This precondition could not be achieved by the operator MOSPLOT-construct-crude-nav-file, because this operator required knowledge of the latitude and longitude pointing location of the spacecraft for an

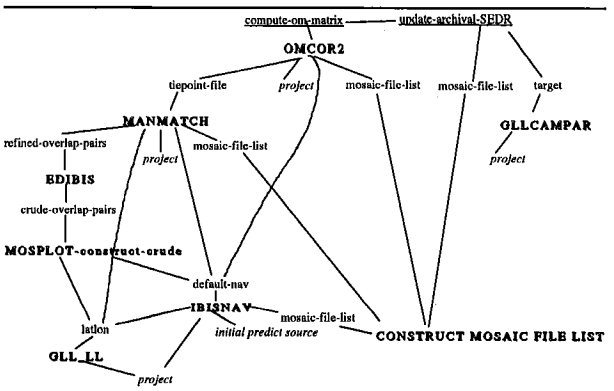

Figure 8: Operator Completion Tool Debugging Trace

image in the file-list. This is where the bug occurred, planning operators existed to extract a single file from the file-list relating tot he operator MOSPLOT-construct-crude-nav-file, however, these planning operators presumed knowledge of the name of the file (e.g., 1126.IMG). However, the planning operator to extract a filename from a sequence of image files resulted in the name of a variable which at runtime would be bound to the name of the file (e.g., &middle-file which at run time would be bound to 1126.IMG). This is due to the fact that the file ordering is run-time dependent. Thus, the GLL_LL operator (which uses spacecraft navigation information to derive the latitude and longitude of an image file) needed to be modified to accept either a variable bound to a file, or the name of the file itself - resolving the bug.

Unfortunately, we have as of yet not been able to determine any good heuristics for controlling the use of these tools in a more tractable way. However, in their current form, the completion analysis tools have proved quite useful in debugging the MVP radiometric correction and color triplet reconstruction knowledge base.

Application Use and Payoff

MVP2.0 is implemented in C and runs Sun SparcStations under Unix and Motif and under VMS on Vaxes. MVP is currently operational and available for use by analysts at JPL's Multimission Image Processing Laboratory (MIPL) for radiometric correction, color triplet reconstruction, and mosaicking with relative or absolute navigation, registration, and simple filtering and stretching tasks. For these tasks MVP reduces effort to generate an initial PDF for an expert analyst from 1/2 a day to 15 minutes and reduces the effort for a novice analyst from several days to 1 hour. Thus, by using MVP an analyst achieves over an order of magnitude improvement in productivity in generating image processing PDFs.

Application Development and Deployment

Initial development on the MVP concept began in August 1992 and a proof of concept demonstration system MVP0 was produced running in LISP on a SUN sparcstation by late September 1992 for basic correction tasks (radiometric correction, missing line fillin, despiking data, photometric correction, blemish removal and reseau removal) for Voyager project images. This demonstration was invoked using a text-based interface and accessed a dummy image database rather than accessing the actual image database. The complete effort for the proof of concept demonstration was approximately 1 work-month.

This prototype was well received by the analysts, and work began on a more complete prototype MVP 1.0. This version was also implemented in LISP and accessed a dummy image database in flatfile format. The domain theory for MVP 1.0 extended MVP0 by including Galileo project images, and added absolute navigation of images as well as several simple filtering and stretching options. MVP1.0 used a graphical user interface running under Openlook/X. MVP 1.0 was completed by March 1993. MVP 1.0 was further extended to cover application programs for registration of data and simple mosaicking tasks in version 1.1, which was completed in September 1993. The complete development effort for MVP 1.0 and 1.1 was approximately 0.9 work-years, which includes analyst time and MIPS programming support.

In October 1993, work began on MVP2.0 which was intended to be an operational system in the Multimission Image Processing Laboratory (MIPL). From an AI planning standpoint, MVP2.0 would be very similar to MVP1.1, with the major difference that it was to be written in C. This included migration of the decomposition planner (to CLIPS) and integration of Flex and Bison to parse input operator and rule files. The majority of the port was completed by March 1994, with April and May being spent on testing, developing a Motif-based GUI, and interfacing MVP2.0 to actual VICAR database access routines. MVP2.0 was installed in MIPL May 1994. During the summer of 1994, MVP2.0 was extensively tested in the operational setting and used to generate image products. Simultaneously, it was extended to cover more complex correction tasks involving registration, and relative navigation. Also, during this period, a number of knowledge base development tools were produced (Chien94b). The complete development effort for MVP 2.0 from September 1993 through September 1994 was approximately 2.2 work years.

Current efforts focus on two fronts: 1) expanding the domain coverage to further image processing tasks ; 2) providing a development environment to facilitate extension to new image processing tasks ; and 3) fielding MVP to a University VICAR image processing site (called Home Institution Image Processing Sites or HIIPS). We are currently working on extending the domain knowledge represented in MVP2.0 to cover more complex mosaicking tasks as well as filtering and stretching tasks. Development environment enhancements include tools to analyze operator sets and rule sets to find simple errors (e.g. typographical errors) that result in domain theories where no actions can achieve a goal. Towards fielding MVP at a HIIPS site, we are currently in contact with personnel from the department of Geology at Arizona State University about a collaborative effort to field MVP for Galileo and Magellan science image processing.

Maintenance

Initial development of the planning knowledge base was performed by LISP programmer AI personnel with significant background in AI planning systems (Versions 1.0 and 1.1). The domain theories for Version 2.0 was developed by a software engineer with little AI background. The current domain theory is being used to describe to analysts the overall process of constructing planning decomposition rules and operators, with the intention that further versions of the domain theory will be developed by analysts or other VICAR users. Towards support of this goal, we have been developing a planning knowledge base debugging environment, which provides static analyses of the planning knowledge base to perform simple checks for achievability of goals and runtime tools to isolate failing preconditions (Chien94b). In current plans, maintenance and extension of the planner and development environment will be supported by AI group personnel for the near future (e.g., 1-2 years) with the intention that maintenance and extension of MVP will eventually be taken over by the Image Processing Section, with the AI group providing continuing support in a consulting role.

Conclusions

This paper has described the application of AI planning techniques to automate image processing. This application has resulted in the fielding of MVP2.0, which reduces the effort to produce radiometric correction, color triplet reconstruction, and mosaicking image processing procedures by over an order of magnitude. MVP2.0 uses a hybrid 'approach to planning, using hierarchical task decomposition and operator-bsed planning paradigms, as well as traditional syntax translation methods. This successful application is being expanded to cover additional areas of image processing and fielding to remote university image processing sites.

References

D. Chapman, "Planning for Conjunctive Goals, 1987, "*Artificial Intelligence 32*, 3.

S. Chien, "Using AI Planning Techniques to Automatically Generate Image Processing Procedures: A Preliminary Report," *Proceedings of the Second International Conference on AI Planning Systems*, Chicago, IL, June 1994, pp. 219-224.

S. Chien, "Towards an Intelligent Planning Knowledge-base Development Environment," *Proceedings of the 1994 AAAI Fall Symposium on Learning and Planning: On to Real Applications*, New Orleans, LA, November 1994, pp. 23-27.

S.Chien,"Automated Synthesis of Image Processing Procedures for a Large-scale Image Database," *Proceedings of the First IEEE International Conference on Image Processing*, Austin, TX, November 1994, Vol. 3, pp. 796-800.

K. Erol, J. Hendler, and D. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning," *Proceedings of the Second International Conference on AI Planning Systems*, Chicago, IL, June 1994, pp. 249-254.

O. Etzioni, "Acquiring Search Control Knowledge via Static Analysis," *Artificial Intelligence*, to appear.

Y. Iwasaki and P. Friedland, "The Concept and Implementation of Skeletal Plans,"*Journal of Automated Reasoning 1, 1 (1985)*, pp. 161-208.

A. Lansky, Localized Planning with Diverse Plan Construction Methods, Technical Report FIA-93-17, NASA Ames Research Center, June 1993.

S. LaVoie, D. Alexander, C. Avis, H. Mortensen, C. Stanley, and L. Wainio, VICAR User's Guide, Version 2, JPL Internal Document D-4186, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 1989.

J. S. Pemberthy and D. S. Weld, "UCPOP: A Sound Complete, Partial Order Planner for ADL," *Proceedings of the Third International Conference on Knowledge Representation and Reasoning* , October 1992.

K. Ryu and K. Irani, "Learning from Goal Interactions in Planning: Goal Stack Analysis and Generalization," *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, 1992, pp. 401-407.

M. Stefik, "Planning with Constraints (MOLGEN: Part 1)," *Artificial Intelligence 16,2(1981)* , pp. 111-140.