Automated Data Processing as an AI Planning Problem

Keith Golden Wanlin Pang[1] Ramakrishna Nemani Petr Votava[2]
NASA Ames Research Center
Moffett Field, CA 94035
keith.golden@nasa.gov

1 Introduction

NASA's vision for Earth Science is to build a "sensor web": an adaptive array of heterogeneous satellites and other sensors that will track important events, such as storms, and provide real-time information about the state of the Earth to a wide variety of customers. Achieving this vision will require automation not only in the scheduling of the observations but also in the processing of the resulting data. To address this need, we have developed a planner-based agent to automatically generate and execute data-flow programs to produce the requested data products. Data processing domains are substantially different from other planning domains that have been explored, and this has led us to substantially different choices in terms of representation and algorithms. We discuss some of these differences and discuss the approach we have adopted.

1.1 TOPS Case Study

As a demonstration of our approach, we are applying our agent, called IMAGEbot, to the Terrestrial Observation and Prediction System (TOPS, http://www.forestry.umt.-edu/ntsg/Projects/TOPS/), an ecological forecasting system that assimilates data from Earth-orbiting satellites and ground weather stations to model and forecast conditions on the surface, such as soil moisture, vegetation growth and plant stress (Nemani *et al.* 2002). Prospective customers of TOPS include scientists, farmers and fire fighters. With such a variety of customers and data sources, there is a strong need for a flexible mechanism for producing the desired data products for the customers, taking into account the information needs of the customer, data availability, deadlines, resource usage (some scientific models take many hours to execute) and constraints based on context (a scientist with a palmtop computer in the field has different display requirements than when sitting at a desk). IMAGEbot provides such a mechanism, accepting goals in the form of descriptions of the desired data products.

The goal of the TOPS system is the monitoring and prediction of changes in key environmental variables. Early warnings of potential changes in these variables, such as soil moisture, snow pack, primary production and stream flow, could enhance our ability to make better socio-economic decisions relating to natural resource management and food production (Nemani *et al.* 2000). The accuracy of such warnings depends on how well the past, present and future conditions of the ecosystem are characterized.

The inputs needed by TOPS include satellite data, such as Fractional Photosynthetically Active Radiation (FPAR), and weather data, such as precipitation. There are several potential candidate data sources for each input required by TOPS at the beginning of each model run. The basic properties of the inputs are listed in Table 1. Even with this fairly small model, there is a good variety of inputs we need to select from, depending on our goal.

In addition to the attributes listed in the table, data sources also vary in terms of quality and availability — some inputs are not always available even though they should be. For example, both the Terra and Aqua satellites have experienced technical difficulties and data dropouts over periods ranging from few hours to several weeks. Depending on the data source, different processing steps are needed to get the data into a common format. We have to convert the point data (CPC and Snotel) to grid data, which by itself is fairly complex and time-consuming, and we must reproject grid data into a common projection, subset the dataset from its original spatial extent and populate the input grid used by the model. The data are then run through the TOPS model, which generates desired outputs. TOPS provides only a simple illustration of the potential problems, and is less complex than many other models and systems in the Earth sciences, some of which take dozens of different inputs, with sizes reaching into terabytes for each model run.

The next section discusses the Data Processing Action Description Language (DPADL), which is used to to describe the complex data structures, constraints and programs that arise in data processing domains. DPADL is an expressive, declarative language with Java-like syntax, which allows for arbitrary constraints and embedded Java code. Section 3 discusses our constraint-based planner, which accepts goals in the form of data descriptions and synthesizes data-flow programs using the DPADL action descriptions. The constraint solver can handle numeric and symbolic constraints, as well as constraints over strings and even arbitrary Java objects. The latter are evaluated by executing the code embedded in constraint definitions, specified in the DPADL input file. Additionally, it can solve a limited class of univer-

[1]QSS Group Inc
[2]California State University, Monterey Bay

Source	Variables	Frequency	Resolution	Coverage
Terra-MODIS	FPAR/LAI	1 day	1km, 500m, 250m	global
Aqua-MODIS	FPAR/LAI	1 day	1km, 500m, 250m	global
AVHRR	FPAR/LAI	10 day	1km	global
SeaWIFS	FPAR/LAI	1 day	1km x 4km	global
DAO	temp, precip, rad, humidity	1 day	1.25 deg x 1.0 deg	global
RUC2	temp, precip, rad, humidity	1 hour	40 km	USA
CPC	temp, precip	1 day	point data	USA
Snotel	temp, precip	1 day	point data	USA
GCIP	radiation	1 day	1/2 deg	continental
NEXRAD	precipitation	1 day	4 km	USA

Table 1: TOPS input data choices

sally quantified constraints (Golden & Frank 2002). Section 4 discusses related work.

2 DPADL Language

In the course of developing IMAGEbot, we found that existing action representation languages were inadequate for describing data processing domains. To address these deficiencies, we developed a new language called DPADL, for Data Processing Action Description Language (Golden 2002). DPADL provides features tailored for data processing domains, such as:

- First-class objects: Most things in the world and in software environments can be viewed as objects with certain attributes and relations to other objects. For example, a file has a name, host, parent directory, owner, etc. Even more importantly, data files often have complex data structures. The language should provide the vocabulary for describing these structures. DPADL is an object-oriented language, with a syntax based on Java and C++.
- Constraints: Determining the appropriate parameters for an action can be challenging. Parameter values can depend on other actions or objects in the plan. The language should provide the ability to specify such constraints where they are needed. DPADL supports built-in and user-defined constraints over any type, including strings and Java objects, and universally quantified constraints over sets.
- Integration with a run-time environment: It is not sufficient to generate plans; it is necessary to execute them, so there must be a way to describe how to execute the operations provided by the environment and obtain information from the environment. The language should allow the specification of "hooks" into the runtime environment, both to obtain information and to initiate operations. DPADL provides these hooks by permitting embedded Java code in definitions of new constraints and methods for executing actions. Variables used in planning and constraint reasoning can reference Java objects as well as primitives such as integers and strings, so fine-grained interaction with the Java runtime environment is possible.
- Object creation and copying: Many programs create new objects, such as files, sometimes by copying or modifying other objects. The language must provide a way of describing such operations. DPADL allows effects to create new objects, which optionally may be declared as copies of existing objects, in which case it is only necessary to list the ways in which the objects differ; all other attributes are inherited from the preexisting object.
- Operations on large or infinite sets: Many programs act on all members of some set. For example, a backup operation acts on all files on a disk and an image processing operation may affect all pixels in an image, in a specified region of an image, or matching a specified criterion. The language should support universal quantification to describe such operations. DPADL provides universally quantified goals and effects, even when the sets quantified over are infinite.

For an illustration of how these language elements interact, we consider the representation of mosaic tiles in DPADL. Many satellites continuously image whatever portion of the Earth they pass over, like giant hand-held scanners. For convenience, the resulting "swath" data is usually reprojected into onto a 2D "map" and chopped up into "tiles," corresponding to a regular grid drawn over the map. To obtain the data pertaining to a particular region of the Earth, we first identify and obtain the tiles that cover that region and then combine them into a single image, known as a mosaic, and crop away the pixels outside the region of interest.

We represent these tiles in the planner as first-class objects. These objects have attributes describing, among other things, the physical measurement the data in the tile represent, the position of the tile on the grid, the projection used to flatten the globe, and the region of the Earth covered by the pixels in the image. One of the "hooks" we provide to the runtime environment is to allow DPADL objects to represent Java objects, which we represent using the keyword **mapsto**. For example,

```
static type Tile extends Object
            mapsto tops.modis.Tile
```

means that the type Tile corresponds to the Java class tops.modis.Tile. The keyword `static` means that Tile is immutable: tiles can be created but never modified.[1] The

[1]This was an unfortunate keyword choice, as it differs from the Java meaning of `static`.

opposite of `static` is `fluent`. The designation of objects (or functions) as static is very useful to the planner, as we discuss in Section 3.5.

We then need to specify how to map from the attributes of `Tile` to the corresponding methods and fields of `tops.modis.Tile`. This is a two-step process. First, we specify some of the attributes as constraints. Then we define the constraints using Java code that refers to the fields and methods of `tops.modis.Tile`. For example, to obtain the `day` attribute of `Tile`, we call the method `getDay()` on the corresponding Java object:

```
unsigned day {
    constraint {
       value(this) = $this.getDay()$;
    }
}
```

The dollar sign is used to delimit embedded Java code. We can also define constraints that go the other way: given some attribute(s) of a `Tile`, return the `Tile`(s) with those attributes. For example, one attribute of a `Tile` is that it `covers` a given longitude-latitude pair. Given a particular longitude and latitude, the constraint solver can invoke a method to find a single tile that covers it, but it can do even better. Given a rectangular region, represented by intervals of longitude and latitude, it can invoke a method to find a set of tiles covering that region.

```
boolean covers(float lon, float lat) {
  constraint {
    ...
    // returns the set of tiles covering
    // a given lon/lat range.
    {this}([lon], [lat], d=day, y=year,
           p=product, value)
      = {$ if(value)
              return tm.getTiles(lon.max,
                                 lat.min,
                                 lon.min,
                                 lat.max,
                                 d, y, p);
            else return null; $};
  }
}
```

The syntax used in this example is unimportant. What it means is "given an interval over longitude, an interval over latitude and single values for the day, year, and product (measurement type), invoking the method `tm.getTiles` will return the set of tiles matching those criteria."

As the above examples illustrate, the universe is very large and incompletely known. Most of the "sensing" needed by the planner can be handled transparently through constraint execution, but because many of the constraints are one-way (e.g., we can obtain the day for a given tile, but not the set of tiles for a given day), the planner needs to be able to reason about information that is not currently known but will become known. This essentially dictates a first-order representation.

In addition to being large and unknown, the universe is dynamic. Most data-processing actions produce new objects, often by copying and modifying existing objects. We describe actions that create new objects using the keyword **new**. To state that one object is obtained by copying another, we use the keyword **copyof**. For example, to describe the effect of an action `reproject` that changes the projection of its input `tileIn` to `newProjection`, we write:

```
new Tile tileOut copyof tileIn {
   projection = newProjection;
}
```

Although `tileOut` is a completely distinct object from tileIn, we do not specify all the attributes of `tileOut`, only those that differ from `tileIn`.

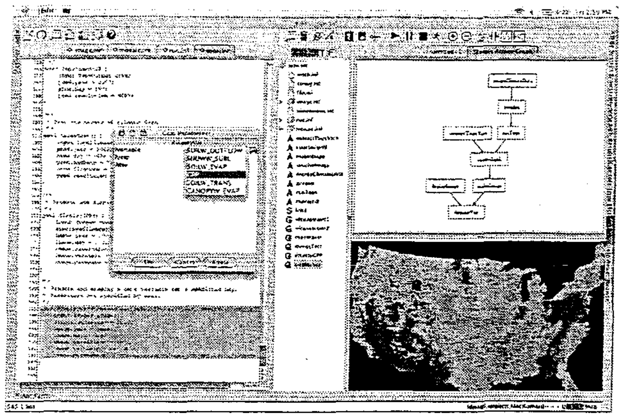


Figure 1: The IMAGEbot development environment, running as a jEdit plugin. The frame on the left shows one of the files comprising the TOPS domain description. The frame on the upper right shows an abstracted view of a plan for the selected goal. More detailed views can also be shown.The frame on the lower right is the data returned by the TOPS server after executing the plan.

3 Planning in the Large

Data processing has traditionally been automated by writing shell scripts. There are some situations when scripts are the best approach: namely, when the same procedure is to be applied repeatedly on different inputs, the environment is fairly stable and there are few choices to be made. However, in many applications, including TOPS, none of these assumptions holds. There are many different data products we would like the system to produce, there are many inputs and data-processing operations to choose from in producing those products, and the availability of these inputs can change over time. Additionally, the domain lends itself to planner-based automation; it has precisely characterized inputs and outputs and operations whose effects can also be precisely characterized. However, there are significant differences between Earth Science data processing and more traditional planning domains, which calls for different techniques. Notable features of data processing domains include large dynamic universes, large plans, incomplete information and uncertainty.

3.1 Decisions, decisions

As we discussed in Section 1.1, we have a number of data sources to choose from, which are applicable under different circumstances. Data sources include several satellites, ground stations, and outputs from other models, forecasts and simulations.

In addition to input choices, we also have several choices of models to use with the data. As with the data, the models produce results of various quality, resolution, and geographic extent. Moreover, there may sometimes be significant trade-offs in performance versus precision. An FPAR/LAI algorithm provides a good example of this trade-off. We can produce an FPAR/LAI pixel using either a lookup table $O(1)$, or a radiative transfer method, $O(n \log n)$ (Knyazikhin *et al.* 1999). The radiative transfer method provides better results, but can take substantially more time. Depending on whether time or accuracy is more important, either method may be preferred.

Another reason for using different models at different times is their possible regional character. Some models are highly specialized and provide very good and precise results in only certain parts of the world. This is partially due to the fact that the scientists who develop these models have a great deal of knowledge about specific geographic area (Pacific Northwest, the Amazons, etc.). They have collected large amounts of local data over the years, and were able to develop models whose outputs are highly accurate in these regions. We usually don't want to use these models when we are concerned with global monitoring, but they are useful when we have identified an important event occurring at the region where we have a very accurate regional model.

3.2 Large dynamic universes

In the last decade, the tide in the planning community has shifted from lifted action representations to ground representations, thanks largely to the success of planners like Graphplan (Blum & Furst 1997) and HSP (Bonet & Geffner 2001) and to the benchmark planning domains made possible by the International Planning Competition. The simple fact is that, at least for these benchmark domains, planners that use ground actions are faster. There has been recent progress (Younes & Simmons 1998) in applying some of the lessons learned from these planners to speed up planners that use lifted actions, but today the fastest planners all use ground actions.

However, there are planning problems for which it is not possible to use ground actions, for example, when not all members of the universe are known at planning time. This is trivially true in information integration domains, such as (Knoblock 1996) and (Etzioni 1996), where the job of the planner is to construct a plan to consult multiple information sources, such as databases or web sites, in order to answer a query. In such domains, virtually no members of the universe may be known to the planner at the time of plan generation.

In data processing domains, too, it is impossible to identify in advance all objects in the universe. Furthermore, most actions create new objects, so the universe is not even static. An examination of the planning problems from the Third International Planning Competition (IPC3) reveals that even the hard problems typically have fewer than 100 objects total. In contrast, if we consider a single product from a single instrument (MODIS) on a single satellite (say, Terra) for a single day, there are 288 tiles. To produce a given data product, we may need to consider multiple products from multiple instruments, residing on multiple satellites, and multiple days' worth of data.

While the details of the specific files to process could be abstracted away in some cases, such an approach is not robust. Particular files may need special processing that other files do not. Sometimes needed files are missing, and substitutes from other sources must be obtained.

Even worse, files are not the smallest unit of granularity; they have sub-structure. For example, image-processing actions act on pixels in the image — either all pixels or a subset determined by some selection criteria. Again, this detail can sometimes be abstracted away, but not always. Additionally, many actions take numeric and string arguments. Appropriate values for these arguments may be determined through constraint reasoning, but there is no way to list all possible values *a priori*. The sheer volume of possible actions makes a grounded representation unsuitable.

Although we cannot use a grounded representation, we would still like to benefit from some of the techniques that have been developed over the past ten years. As we discuss in Section 3.5, we adopt a lifted variant of a relaxed plan-graph analysis, combined with a constraint-based search.

3.3 Large plans

Large universes, combined with universal quantification, can lead to large plans. Earth-Science data processing is very data intensive; producing a single data product can easily require processing hundreds or thousands of files. However, complexity need not grow according to plan size. Whereas traditional benchmark planning problems involve a lot of interactions, making the difficulty of planning exponential in the size of the plans produced, data-processing domains are "embarrassingly parallel." Except for competition for resources such as memory and CPU, the processing required for one mosaic tile does not interfere with the processing for another tile. Indeed, even operations on individual pixels tend to be independent of operations on adjacent pixels. This parallelism is manifest in the structure of the data-flow plans, which tend to be shallow but bushy, with many instances of the same actions operating on different inputs. Even though actions do not directly interfere with each other, there may be constraints between parameter values that arise when planning with a lifted representation. The sub-problems corresponding to the parallel branches of the plan are not truly independent until values for these shared variables have been chosen. Making these choices early thus increases parallelism, at the possible cost of premature commitment.

We have an even more powerful weapon to combat complexity: the use of first-order representations at every stage of the planning process, including constraint reasoning. Given an "embarrassingly" parallel planning problem, it is often possible to construct a very compact plan with simple loops that iterate over, say, all tiles matching a given set of criteria. To facilitate the detection of independence among actions and subgoals, we exploit the fact that certain types are labeled as "static," or immutable. Detecting independence among actions that produce only static objects is trivial — unless one directly or indirectly supports the other, they are independent.

3.4 Incomplete information and uncertainty

There has been considerable work in planning under incomplete information and uncertainty. However, it is worthwhile to compare and contrast data processing domains with other domains that involve incomplete information. In classical planning domains that involve uncertainty and sensing, such as the infamous bomb-in-the-toilet domain, all possible worlds are explicitly enumerated, which facilitates the case analysis necessary to solve these problems. Enumerating just the objects in a single world, let alone all possible worlds, is infeasible in data processing domains. Even the knowledge-based representation of (Petrick 2002), which does not rely on enumerating possible worlds, assumes the objects in the universe are fixed and constant across all possible worlds. In fact, the objects in the universe are not even constant for a single world – over time, objects are both created and destroyed. We adopt the Local Closed-World (LCW) reasoning introduced in (Etzioni, Golden, & Weld 1997) to efficiently reason about incomplete information in the face of very large universes. In IMAGEbot, we deal with three different kinds of uncertainty, and each is handled differently:

- Unknown information that must be known by the agent in order to complete the plan: For example, the information may be used to provide the value of a variable, or select among alternative actions. This information is sensed, not through explicit sensing actions but through the evaluation of constraints, which in turn causes code to be executed to obtain the correct values. For example, if we want to know the mosaic tiles providing a given measurement for a particular region, we can evaluate the constraint associated with the relation *tile*.covers(*lon*, *lat*) for specified intervals of *lon* and *lat*. That, in turn, causes the Java method getTiles to be called, which connects to the TOPS sever to obtain the appropriate set of tiles. In contrast to (Golden 1998), this approach cannot handle sensing actions with preconditions, because the constraints are always applicable, limited only by knowledge of the relevant variable domains. On the other hand, it affords great versatility in the manner in which information is gathered.
- Unknown information that need not be known by the agent in order to complete the plan: For example, if the user requests a file that contains gridded evening temperature values for Montana at 8 km resolution, and the agent has gridded temperatures for the western US at 1 km resolution, it need only select the appropriate subset of the data and reduce the resolution. Even though the agent never knows what the actual temperature values are, it can be confident that the file it returns to the user contains the requested information. In this sense, it is analogous to conformant planning, i.e., producing a plan that is guaranteed to work in any possible state of the world, without knowing the actual state. In fact, the metadata reasoning that the planner employs is remarkably similar to the case analysis employed by conformant/contingent planners. In order to represent that a data file contains specific information, such as temperatures, we rely on metadata formulas (Golden 2000), first-order descriptions of information sources that describe data contents in terms of the information about the world contained in the data.
- Uncertainty in how well the values stored in the data files represent the physical variables they are supposed to represent: Although it is tempting to represent these uncertainties in terms of probability distributions, the probabilities are unknown, even to the scientists who are experts in the field. Instead, we represent these uncertainties in an ad hoc manner, in terms of "data quality." *A priori* quality values can be assigned to data from different sources, modified by information known about specific data files. For example, satellite data have quality assurance flags, reporting problems such as cloud cover, "dead detectors," and values that are outside the expected bounds. Additionally, various processing operations can affect data quality, which we can express in terms of a mathematical relationship between the quality of the input and the quality of the output.

3.5 Planning approach

Planning is a two-stage process. The first stage consists of a Graphplan-style reachability analysis, (Blum & Furst 1997) used to derive heuristic distance estimates for the second stage, a constraint-based search. These stages are not entirely separate, however; constraint propagation occurs even in the the graph-construction stage, and the graph is refined during the constraint-search phase.

Lifted planning graphs Although our planning graph approach is inspired by Graphplan, there are a number of significant differences that arise from the nature of the planning domains we are interested in:

- Nodes in the graph are lifted, and each node may represent a *set* of actions or propositions of a given type. For example, in a given layer of the graph, there might be a single node corresponding to the action schema "compress(*file*)," for a thousand different instances of *file*. Nodes may be split when doing so would improve the reachability analysis, and they may, in some cases, be grounded, but in general there is not a one-to-one correspondence between nodes in the graph and actions in the final plan.
- Instead of the propositions of Graphplan, we have *literals* (lifted propositions), first-order *metadata* formulas and *objects*, described in Section 2. DPADL objects can be (and usually are) static or *immutable*, meaning they can be created but never modified. From the planner's perspective, this is very important to know, because all preconditions associated with a given object must be satisfied by effects of the same action. We cannot have one action that creates a tile and another that changes the tile's projection; to obtain a different projection, the agent must produce a new tile. From the standpoint of the planning graph, there

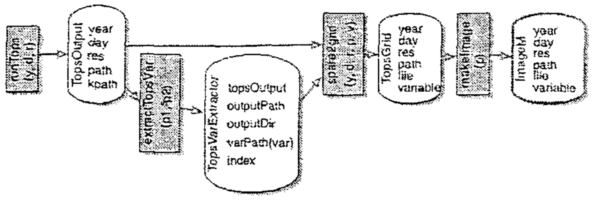


Figure 2: A portion of the planning graph corresponding to a plan from the TOPS domain (Figure 1). The rectangular nodes represent actions and the round nodes represent objects that are input/output by the actions. The labels inside object nodes are attributes of the objects.

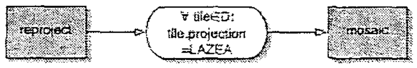
are two important implications: First, there is no point in having separate nodes for the individual attributes of an immutable object, since all attributes must be supported by the same action. Second, no two actions can produce the same output, so the effects an action has on its output(s) can never interfere with the effects of other actions (i.e., there are no mutexes). These restrictions don't apply to the attributes of mutable objects.

- Instead of Graphplan-style binary mutexes between actions or conditions, we have arbitrary constraints and a constraint propagation algorithm tailored to the lifted planning graph representation. Constraint propagation serves two purposes: to reduce the set of ground actions and conditions represented by nodes in the graph by reducing the domains of their variables and to eliminate nodes and arcs in the graph completely by detecting inconsistent constraints.
- The initial graph construction is backward from the goal, to avoid adding irrelevant actions. Afterward, variable bindings are propagated forward from the initial state, and unreachable nodes are eliminated.

A portion of a lifted planning graph with object nodes is shown in figure 2.

Constraint generation After the graph is constructed, heuristic distance estimates for guiding the search are computed, and a constraint network representing the search space is incrementally built. It is incremental because the planning graph comprises a compact representation of the search space, in which each action node can represent multiple concrete actions in the final plan. Since the number of possible actions can be large, even infinite, we cannot simply generate all of them at once but do so lazily during search. This is handled using a dynamic CSP (DCSP), in which new variables and constraints can be added for each new action and causal link in the plan.

However, it is not always necessary to ground out all of the actions in the CSP. As discussed in (Golden & Frank 2002), our constraint reasoning techniques can handle constraints that include universally quantified variables, and we use this to our advantage when solving universally quantified goals or preconditions in domains that are highly symmetric. For example, consider the subgoal of constructing a mosaic, using the LAZEA projection, from all tiles covering the continental US for a given day and satellite data product. A precondition of the `mosaic` action will be that the input tiles are all in the LAZEA projection. This precondition can be satisfied for any given tile by using the `reproject` action. The representation of this in the planning graph will look like:



Although the planner could expand the reproject node during planning time, creating one new copy of the action for each tile to be reprojected, in this case it won't do so. Rather, it will treat the action as a "loop," iterating over all of the tiles in a set. Since none of the actions interfere with each other, and none of the tiles requires special treatment, the plan for each tile is the same. The constraints generated for this plan will be universally quantified, and the constraint reasoning system will attempt to prove that the quantified constraints are satisfied for all members of the domain.

Just as one node in the graph can represent multiple concrete actions, one concrete action can be represented by multiple nodes. That is, two nodes in the graph might actually *unify*. We don't commit at planning time to whether a condition is supported by a new or existing action, and having two separate nodes for a given action schema (or two action variables in the constraint network) does not necessarily mean that there are two distinct instances of that schema in the plan. Similarly, two object variables may both designate the same object. The burden this least commitment approach imposes on the constraint network is an additional $O(n^2)$ constraints for every set of n variables that could conceivably unify. For example, suppose we have two action variables a_1 and a_2, both representing instances of `reproject`, which has one output. We will represent the output variables of a_1 and a_2 as $a_1.out$ and $a_2.out$, respectively. Since two distinct actions cannot have the same output, if both outputs variables are forced to codesignate (for example, because each is constrained to be the sole input of a single concrete action), then the actions variables must also codesignate: $(a_1.out = a_2.out) \Rightarrow (a_1 = a_2)$. Similarly, if the actions codesignate, then their corresponding inputs and parameters must also codesignate. We are exploring an alternative representation of these constraints that avoids explicitly generating all $O(n^2)$ constraints.

The constraints generated for a given planning problem are simply a naive translation of explanatory frame axioms corresponding to the planning problem. We have boolean variables for all of the arcs (causal links) and conditions in the plan. For each condition c, we have a constraint specifying that exactly one of possible causal links l_i supporting that condition is chosen:

$$\text{ImpliesXOR}(c, l_1, \ldots, l_n),\ i.e.,\ c \Rightarrow l_1 \otimes \ldots \otimes l_n$$

For each link l and each condition c_k that the link can support (a link can support multiple conditions when each condition is an attribute of an immutable object), we have a constraint stating that if l is chosen, then c_k is true iff a condition $ant(l, c_k)$, obtained by regressing c_k through the action, is true.

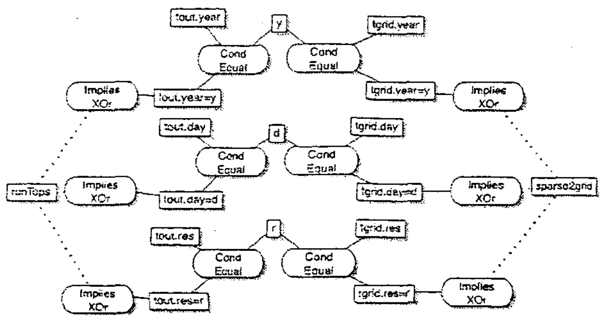


Figure 3: A portion of the constraint network from the planning graph shown in Figure 2.

$$\text{ImpliesEqual}(l, ant(l,c_k), c_k)\text{: } l \Rightarrow ant(l,c_k) = c_k.$$

Conditions such as $ant(l,c_k)$, which is obtained by goal regression, may correspond to fairly complex expressions. These are represented in a very straightforward manner. For example, given the expression $x = y \vee (x = 1 \wedge y = 2)$, we introduce new boolean variables v_{or}, v_{and}, v_{eq}, v_1, v_2, and the following constraints

$$\begin{aligned}
&\text{CondOr}(v_{or}, v_{eq}, v_{and}),\ \textit{i.e.},\ v_{or} \Leftrightarrow v_{eq} \vee v_{and}\\
&\text{CondAnd}(v_{and}, v_1, v_2),\ \textit{i.e.},\ v_{and} \Leftrightarrow v_1 \wedge v_2\\
&\text{CondEqual}(v_{eq}, x, y),\ \textit{i.e.},\ v_{eq} \Leftrightarrow x = y\\
&\text{CondEqual}(v_1, x, 1),\ \textit{i.e.},\ v_1 \Leftrightarrow x = 1\\
&\text{CondEqual}(v_2, y, 2),\ \textit{i.e.},\ v_2 \Leftrightarrow y = 2
\end{aligned}$$

Some of the constraints from the plan fragment shown in Figure 2 are illustrated in Figure 3. Clearly, this can result in a considerable number of variables and constraints. Because we are generating the constraints from a STRIPS-like representation, in contrast to the Europa planner (Jönsson *et al.* 2000), in which the domain modeler specifies the constraints for the explanatory frame axioms directly, the resulting constraint network is much less concise than it could be. The impact on search is generally not a problem, since the constraint search is strongly informed by the structure of the plan, though when the variable-ordering heuristics do fail, they can fail badly. More of an issue is that it can be difficult for a domain developer to make sense of the constraints. We are working on tools to make this job easier, and also exploring ways to prune down the number of generated constraints.

The large number of boolean variables and disjunctions can effectively put a brake on constraint propagation. In some cases, this is desirable, as it focuses the propagation on parts of the search space that are being explored. However, it can also result in a rather myopic view of the CSP, where we would prefer a more global view. Our graph-based propagation algorithm effectively gets around this problem by propagating the domains of "interesting" variables along the structure of the planning graph, unioning the results when it encounters a disjunction and intersecting the results when it encounters a conjunction.

3.6 Constraint reasoning

Many algorithms and systems have been developed for solving constraint problems, ranging from simple backtracking search algorithms to sophisticated hybrid methods. However, constraint networks with infinite domains represent new challenges. In terms of representation, constraints can no longer be represented extensionally as relational tables. It is impossible to store in a computer a relation with infinite entries. From a reasoning point of view, the conventional search algorithms and consistency techniques cannot be applied directly. There is no way to enumerate values of an infinite domain exhaustively. It is unknown to us whether there is a general framework available to represent and to solve infinite constraints problems.

As discussed in Section 3, planner variables, even universally quantified variables, can have infinite domains. Since these variables can appear in constraints, we have implemented a constraint network component capable of solving a class of constraint problems with infinite domains, that is, universally quantified constraints obtained from subgoals of the planner (Golden & Frank 2002). Each variable is associated with a domain. A variable domain can be finite or infinite, in which case it is represented as an interval (for numeric type variables), a regular expression (for string type), or symbolic sets (for object type). The use of regular expressions to represent string domains (Golden & Pang 2003), and the support for universally quantified constraints are both novel, if somewhat unorthodox, contributions to constraint reasoning. The planning-graph-based constraint propagation algorithm, described briefly above, is also novel. The overall algorithm for solving dynamic CSPs is based on the constraint solver used in the Europa planner (Jönsson *et al.* 2000).

4 Conclusions

We have discussed a novel class of planning domains, data-processing domains, that pose a number of challenges for planners, including large dynamic domains, complex data structures and complex constraints. In answer to these challenges, we have introduced the DPADL language, which can represent complex, nested data structures, arbitrary constraints, object creation and copying and fine-scale integration with the Java runtime environment. We also introduced a novel planer, called DoPPLER, which constructs dataflow programs to produce data products satisfying user requests. Novel features of DoPPLER include a lifted variant of the planning graph and a constraint solver that can propagate domain values within a planning graph and can handle universally quantified constraints. For a more detailed discussion of the DPADL language or the constraint solver used by DoPPLER, see (Golden 2003; Golden & Frank 2002; Golden & Pang 2003).

One challenge posed by these domains that we have not yet adequately addressed is the multi-criteria optimization problem inherent in the tradeoff among features such as time, resource consumption and data quality. DoPPLER is currently quite greedy, but the next version will rely on branch and bound, guided by bounds on variable domains provided by our planning-graph-based constraint propagation algorithm.

DoPPLER is not a domain-dependent planner, but it is specialized for a *class* of planning domains: data-processing domains. Even representing approximations of such domains in a language such as PDDL, which assumes a closed, static world, is a considerable challenge (Golden 2003), and propositional planning in such domains is unthinkable. Although all of the novel features of the planner were introduced with data-processing domains in mind, they may prove useful in other domains with similar characteristics. For example, our planning graph approach may be useful for logistics planning. Exploring these issues is the subject of future work.

4.1 Related Work

There has been little work in planner-based automation of data processing. Two notable exceptions are Collage (Lansky & Philpot 1993) and MVP (Chien *et al.* 1997). Both of these planners were designed to provide assistance with data analysis tasks, in which a human was in the loop, directing the planner. In contrast, the data processing in TOPS must be entirely automated; there is simply too much data for human interaction to be practical.

(Blythe *et al.* 2003) addresses workflow planning for computation grids, a similar problem to ours, though their focus is on mapping pre-specified workflows onto a specific grid environment, whereas our focus is on generating the workflows.

Planning for data-processing shares many characteristics with planning for information integration and planner-based software agents (Golden 1998). The primary difference is the need in data-processing plans to reason about information that will never be known to the agent but is nonetheless essential to the task at hand — namely, the information contained in the data files that the agent must process.

The Amphion system (Stickel *et al.* 1994) was designed to construct programs consisting of calls to elements of a software library. Amphion is supported by a first-order theorem prover. The task of assembling a sequence of image processing commands is similar to the task Amphion was designed to solve. However, the underlying representation we use is a subset of first-order logic, enabling the use of less powerful reasoning systems. The planning problem we address is considerably easier than general program synthesis in that action descriptions are not expressive enough to describe arbitrary program elements, and the plans themselves do not contain loops or conditionals.

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *J. Artificial Intelligence* 90(1–2):281–300.

Blythe, J.; Deelman, E.; Gil, Y.; Kesselman, C.; Agarwal, A.; Mehta, G.; and Vahi, K. 2003. The role of planning in grid computing. In *Proc. 13th Intl. Conf. on Automated Planning and Scheduling (ICAPS)*.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Aritificial Intelligence* 129(1–2):5–33.

Chien, S.; Fisher, F.; Lo, E.; Mortensen, H.; and Greeley, R. 1997. Using artificial intelligence planning to automate science data analysis for large image database. In *Proc. 1997 Conference on Knowledge Discovery and Data Mining*.

Etzioni, O.; Golden, K.; and Weld, D. 1997. Sound and efficient closed-world reasoning for planning. *J. Artificial Intelligence* 89(1–2):113–148.

Etzioni, O. 1996. Moving up the information food chain: softbots as information carnivores. In *Proc. 13th Nat. Conf. AI*.

Golden, K., and Frank, J. 2002. Universal quantification in a constraint-based planner. In *Proc. 6th Intl. Conf. AI Planning Systems*.

Golden, K., and Pang, W. 2003. Constraint reasoning over strings. In *Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming*.

Golden, K. 1998. Leap before you look: Information gathering in the PUCCINI planner. In *Proc. 4th Intl. Conf. AI Planning Systems*.

Golden, K. 2000. Acting on information: a plan language for manipulating data. In *Proceedings of the 2nd NASA Intl. Planning and Scheduling workshop*, 28–33. Published as NASA Conference Proceedings NASA/CP-2000-209590.

Golden, K. 2002. DPADL: An action language for data processing domains. In *Proceedings of the 3rd NASA Intl. Planning and Scheduling workshop*, 28–33. to appear.

Golden, K. 2003. An domain description language data processing. In *ICAPS 2003 Workshop on the Future of PDDL*.

Jönsson, A.; Morris, P.; Muscettola, N.; and Rajan, K. 2000. Planning in interplanetary space: Theory and practice.

Knoblock, C. 1996. Building a planner for information gathering: A report from the trenches. In *Proc. 3rd Intl. Conf. AI Planning Systems*.

Knyazikhin, Y.; Glassy, J.; Privette, J. L.; Tian, Y.; Lotsch, A.; Zhang, Y.; Wang, Y.; Morisette, J. T.; Votava, P.; Myneni, R. B.; Nemani, R. R.; and Running, S. W. 1999. MODIS Leaf Area Index (LAI) And Fraction Of Photosynthetically Active Radiation Absorbed By Vegetation (FPAR) Product (MOD15) Algorithm Theoretical Basis Document. http://eospso.gsfc.nasa.gov/atbd/modistables.html.

Lansky, A. L., and Philpot, A. G. 1993. AI-based planning for data analysis tasks. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications (CAIA-93)*.

Nemani, R.; White, M.; Votava, P.; Glassy, J.; Roads, J.; and Running, S. 2000. Biospheric forecast system for natural resource management. In *Procceedings of GIS/EM -4*.

Nemani, R.; Votava, P.; Roads, J.; White, M.; Thornton, P.; and Coughlan, J. 2002. Terrestrial observation and predition system: Integration of satellite and surface weather observations with ecosystem models. In *Proceedings of the 2002 International Geoscience and Remote Sensing Symposium (IGARSS)*.

Petrick, R., B. F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. 6th Intl. Conf. AI Planning Systems*.

Stickel, M.; Waldinger, R.; Lowry, M.; Pressburger, T.; and Underwood, I. 1994. Deductive composition of astronomical software from subroutine libraries. In *Proceedings of the 12th Conference on Automated Deduction*.

Younes, H., and Simmons, R. 1998. On the role of ground actions in refinement planning. In *Proc. 6th Intl. Conf. AI Planning Systems*, 54–61.