In Coenen, Preece & Macintosh (eds) Research and Development in Intelligent Systems, XX (Proocedings of AI-2003), pp.295-308. Springer.

Model-based Planning in Physical domains using SetGraphs*

Max Garagnani
Department of Computing, The Open University
Milton Keynes, UK

Abstract

This paper proposes a non-propositional representation framework for planning in physical domains. Physical planning problems involve identifying a correct sequence (plan) of object manipulations, transformations and spatial rearrangements to achieve an assigned goal. The problem of the *ramification* of action effects causes most current (propositional) planning languages to have inefficient encodings of physical domains. A simpler and more efficient representation is proposed, in which actions, goals and world state are modelled using 'setGraphs'. A setGraph is an abstract data-structure able to capture *implicitly* the structural and topological *constraints* of a physical domain. Despite being *model-based*, the representation also allows the use of *types* and propositional furmulæ to specify additional domain constraints. Experimental results obtained with a specific implementation of the representation indicate significant improvements in performance in all of the domains considered.

1 Introduction

Research in AI and cognitive psychology has since long demonstrated that the type of problem *representation* adopted plays a fundamental role in determining the difficulty of problem solving [4][5], and has identified at least two main types of approaches: a so-called *model-based* (or analogical) representation, and a *propositional* (or Fregean, or sentential) representation [7]. In a model-based representation, facts are represented using a data structure which is, to a significant extent, *isomorphic* to (i.e., a model of) the *semantics* of the problem domain. By contrast, propositional representations (e.g., predicate calculus, modal logic) are *syntactical* descriptions whose structure has no bearing to the structure and semantics of the represented state of affairs [6].

Throughout their history, AI planning domain-description languages have encoded *all* aspects of a problem (and, in particular, *spatial* and *topological* relations) using exclusively propositional representations (e.g., STRIPS, ADL, PDDL [2]). These languages require even the most basic physical *constraints* of the world (such as the fact that an object cannot be simultaneously in two places) to be declared and/or dealt with *explicitly.* In such representations, taking into account the *ramifications* of the effects of an action in domains

*This work was partially supported by the UK Engineering and Physical Sciences Research Council, grant no. GR/R53432/01.

that involve moving and transforming (possibly large) sets of objects – subject to various constraints, physical properties and spatial relations – leads to significant computational-workload overheads.

In spite of various recent extensions in the expressive power of domain-modelling languages (e.g., see [2]), the possibility of using model-based representations to improve planning performance has been left almost completely unexplored. This lack of attention may be attributed to the fact that analogical representations tend to be less expressive and more domain-specific than propositional ones. Nevertheless, although more general, propositional representations are much less efficient, as they lack a semantic (or heuristic) guidance, necessary to effectively navigate large search spaces [6].

The overall aim of this work is to investigate new and expressive domain-modelling formalisms in which the basic, common-sense physical constraints of the world are *implicitly* and effectively encoded in the problem representation. In particular, this paper proposes an abstract representation in which the state of the world is described as a 'setGraph', a data-structure designed specifically to capture the structural and topological constraints of a domain. Although model-based, the abstract character of the representation and its capability to make use of propositional descriptions endow the model with sufficient generality and expressive power to encode a large class of problems.

In the proposed representation, a domain is described through basic concepts of graph and set theory. A computational implementation of the model is, therefore, quite straightforward. Nevertheless, it should be clarified that this paper is not concerned with the definition of a specific language for the computational encoding of the proposed representation.[1] Indeed, the description of the model is kept intentionally abstract and intuitive, in order to avoid any low-level, syntactical details which could commit the representation to a particular language or paradigm.

The rest of the paper is organised as follows: Section 2 identifies more precisely the class of physical domains considered; Section 3 describes the abstract structure of a setGraph, while Section 4 illustrates the meaning of *action* in it and some of its formal properties. Experimental results obtained with a prototype planner implementing an instance of the abstract model are reported in Section 5. Section 6 contains a brief survey of the related literature, and Section 7 concludes by discussing advantages, limitations and possible extensions.

2 Move and Change

In the mid seventies, Hayes and Simon [4] analyzed the performance of subjects solving various 'isomorphs' of the Tower of Hanoi (ToH) problem. These isomorphs used different 'cover stories' but always involved a problem space *identical* to that of a three-disk ToH problem. The rules for state transformation were also identical in number, relevance, and restrictiveness to those of the

[1]The syntax of a domain-description language developed for a particular instance of this representation was proposed in [3].

ToH problem. In spite of the structural identity of the domains, the subjects

> "exhibited large and systematic differences in the relative difficulty that they experienced with two broad classes of isomorphs. These two classes have been labeled *Move* [...] and *Change* problems, respectively [...], on the basis of whether successive problem transformations require moving an object from one spatial location to another, or changing some properties of an object that remains at a fixed location" [5, pp.249-250]

Following the classification proposed by Hayes and Simon, '*move*' domains are defined here as problems involving the *rearrangement* – subject to a set of *constraints* – of a finite number of mobile objects over a finite set of possible locations. By contrast, '*change*' domains are characterised by a set of *stationary* objects that can change some of their properties (subject to a set of constraints) while remaining at fixed locations. These two classes can be thought of as representing the two extremes of a *continuum* of planning domains, in which change of state and movement are present in various degrees. The Tower of Hanoi is a prototypical example of the class of move domains, as it does not contain any change component. Other classical domains that can be included in this category are Blocksworld (BW), Eight-puzzle, Briefcase, Robot world, Gripper, and Logistics. By contrast, the "Monster Change" problem (see Appendix A), one of the ToH isomorphs that Hayes and Simon used in their experiments, can be taken as a prototype of the class of change domains. Another example of a typical change domain is the *Homeowner* domain [11].

The following sections describe a *unified* representation for move and change domains based on 'setGraphs'. The two main ideas motivating the use of setGraphs are the following: (1) a model-based formalism allows a more efficient representation of *any* move problem; and (2) all change problems can be reformulated as *equivalent* move problems. This allows a single and more *efficient* formalism for planning in any domain that falls within the move-change continuum. Point (2) is discussed in Section 4.2, while point (1) is supported by experimental evidence (see Section 5). First of all, however, it is necessary to clarify some of the terms and concepts used throughout the discussion.

2.1 Terminology

The definitions of the concepts of mobile and stationary objects are quite intuitive: an object is 'mobile' if the set of possible actions allows changing its 'location' (i.e., the *place* in which the object lies – see below), and 'stationary' otherwise. A similar definition has been proposed by Long and Fox: "A mobile object is defined to be one that can make a self-propelled transition from being situated in one location to being situated in another location" [9].

Intuitively, a 'location' can be thought of as a *qualitatively* distinct area of space (containing a set of – mobile or stationary – objects). However, one can define a location simply as a type of possibly mobile object that can be *associ-*

ated with a set of objects. Hence, locations are not necessarily stationary.[2] The locations between which a location-object can move can be seen, in turn, as (higher-level) locations. In summary, the representation can be simply limited to *stationary* and *mobile* objects that may be allowed to contain (or be associated with)[3] other objects. Objects that *are* allowed to contain other objects will be called *places*. *Stationary* places represent 'fixed' locations.

For example, consider the Blocksworld domain. Here, each block can be considered a mobile object, able to 'contain' other objects, namely, the block(s) lying on top of it. The predicate `On(x,y)` can be seen as specifying the location of object x (i.e., the content of *place* y). The '`Table`' object can be thought of as a (stationary) place, able to contain blocks. A block can be moved from stack to stack subject to the requirements (*constraints*) that the block itself and its destination (another block) are both 'empty' (i.e., clear).

As illustrated by the previous example, a physical domain can be subject to several (qualitative or quantitative) constraints, which restrict the movement and transformation of the objects and determine the topology and properties of the domain. For example, given two places "*a*" and "*b*", the transition of a mobile object from "*a*" to "*b*" may be subject to the existence of a specific relationship (a 'road') between them, or to the number of objects already in "*b*" being less than a certain value.

Finally, the concepts of 'mobile' and 'stationary' objects should not be confused with those of *dynamic* and *static* objects. Dynamic objects may change their *properties* (value of their attributes) and relations during plan execution, whereas *static* objects cannot. In this respect, stationary objects can be dynamic. In the Grid domain, for example, the state of a (stationary) location does not change as the result of the arrival or departure of an 'agent', but it does change (from 'locked' to 'unlocked') if the agent uses an appropriate 'key'.

3 SetGraphs

In the proposed paradigm of representation, the current state of the world is modelled by a *setGraph*, defined as follows:

Definition 3.1 A *setGraph* is a pair $\langle P, R \rangle$, where P is a finite set of *setNodes*, and R is a finite set of *edges*. N is a 'setNode' *iff* it satisfies one of the two following conditions:

1) $N = \emptyset$

2) $\forall x \in N$, x is a setNode

An 'edge' consists of a pair (x, y), where x, y can be any two of the setNodes appearing in P.

[2]Note that if a location 'A' moves, all objects in A move with it, but the location *of the objects* (their 'container') remains 'A'.

[3]The significance of the different use of the term 'contain' *vs.* 'associated-with' is clarified in Section 3.

SetNodes and edges may be assigned (not necessarily unique) labels. According to the definition, setNodes are 'nested' sets of (possibly labelled) empty sets, with no limit on the level of nesting. SetNodes and edges can be of different *types* (or sorts), and have different pre-defined structure and behaviour. The specification of setNode and edge properties corresponds to the imposition of further constraints on the problem. In particular, the different types of setNodes and edges are organized in two separate *hierarchies*, in which the properties of a type are inherited by all of its instances and sub-types. SetNode types are further divided into two separate hierarchies, namely, a *place* hierarchy and an *object* hierarchy. SetNodes that are instances of place types will be allowed to contain other setNodes (of select types), whereas instances of object types are *necessarily empty* setNodes (i.e, unable to contain other setNodes)[4].

For example, consider a 'Ferry' domain with three cars ("A", "B" and "C"), one ferry ("Ferry") and two locations ("Port1" and "Port2") between which the ferry can sail. The initial state {`(at-ferry Port1) (at A Port1) (at B Port1) (on C Ferry)`} can be encoded as a setGraph $\alpha = \langle P_1, R_1 \rangle$, where:

$$P_1 = \{\ \texttt{Port1\{A\{ \},B\{ \},F\{ \}\}},\ \texttt{Ferry\{C\{ \}\}},\ \texttt{Port2\{ \}}\ \}$$
$$R_1 = \{\ (\texttt{Port1,Port2}),\ (\texttt{Port2,Port1}),\ (\texttt{F,Ferry})\ \}$$

P_1 contains all the setNodes (objects or places) of the domain. `A, B, C` and `F` will have been declared as *objects* (i.e., necessarily empty setNodes) of two types: '`car`' (instances `A, B, C`) and '`ferry_symbol`' (instance `F`). The domain will also have two types of *place* ('`port`' and '`ferry_place`', with instances '`Port1`', '`Port2`' and '`Ferry`', respectively). Finally, the type constraints imposed should allow a '`port`' place to contain any type of *object* but no places, and a '`ferry_place`' to contain ($\leq k$) objects of type '`car`'.

In R_1, two (unlabelled) edges are used to represent a (bi-directional) relationship between the two ports. The edge (`F,Ferry`) encodes an *association* between the ferry-symbol '`F`' and the ferry-place '`Ferry`'. The setNode '`F`' is a symbolic representation of the '`Ferry`' place, and metaphorically contains the set of cars currently on board. Figure 1 depicts a graphical representation of setGraph α, where places are depicted as dashed ellipses and edges as arrows, bi-directional arrows are arcs, and small circles indicate objects.

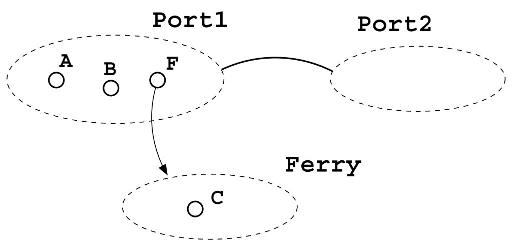

Figure 1: Graphical representation of setGraph $\alpha = \langle P_1, R_1 \rangle$

[4]This does not prevent setNodes from being able to be *associated* to other setNodes.

4 Dynamics of a SetGraph

The possible transformations of a setGraph have been assumed to follow two general, intuitive rules: (1) any setNode can be moved from any (setNode) place to any other place (subject to the specific constraints of the domain); and (2) when a setNode is moved, all edges connecting it to other setNodes remain attached to it, and all setNodes contained in it move with it, while the rest of the state remains unchanged (default persistence).

These rules capture the properties of many physical domains, in which a 'structured' set of places and connecting edges form an underlying *static data structure* that cannot be affected by the possible actions. However, in principle, edges between objects and places could be modified by some of the possible actions of the domain. For example, in a hypothetical '*Road maintenance*' domain, 'off-road' connections between different places could be dynamically *created* or *removed* during plan execution, in order to allow the workers to have access to different locations (and resources) during different phases of the work. In order to model these dynamics, the representation should allow the definition of a set of *primitive* transformations which can modify the underlying structure of the setGraph by adding and removing to/from it edges and setNodes. This type of transformations involves *non-conservative* changes of the setGraph, i.e., actions which do not preserve the initial number of setNodes and edges. These dynamics would appear to be particularly relevant for modelling *production* and *consumption* of resources.

In this paper we concentrate on the representation of actions involving only *conservative* changes; however, the formalism can be easily extended to include such 'lower-level', primitive setGraph transformations.

In many domains, the two general conservative rules introduced earlier are not sufficient to prevent the generation of incorrect moves. For example, in the Ferry domain, a car currently in '`Port1`' could be moved directly to '`Port2`' (or vice versa). In order to prevent undesirable moves, the legal transformations of a setGraph will be allowed to be defined through a set of domain-specific *action schemata*.

4.1 Syntax and Semantics of Action Schemata

An action schema (or *operator*) specifies a transformation of a sub-part of a setGraph. It consists of *preconditions* (specifying the situation required to hold in the relevant elements before the action is executed) and *effects* (describing the changes that its application produces on the elements identified).

The preconditions of an operator will be expressed simply as a *typed* setGraph, in which places, objects and edges are identified by their sort[5]. This allows the schema to be generally applicable in different situations. In addition, the preconditions may be augmented with a set of (qualitative or quantitative) *constraints* on the properties of the elements involved (expressed using a *propositional* formalism).

[5]The use of specific instances in an action should still be permitted, though.

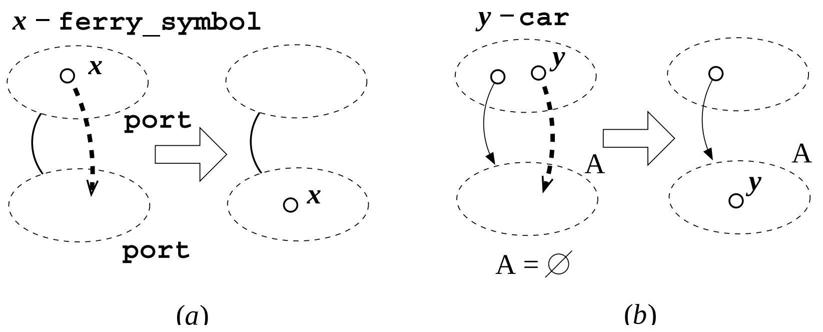

Figure 2: Ferry domain: (a)*Sail* and (b)*Board* operators

The effects of the action application on the current state will be described here using both *procedural* representations (indicating *which* setNodes are being moved to which places) and *declarative* representations (specifying the *resulting* situation in the relevant places after the action application). For example, Figures 2.(a) and 2.(b) depict, respectively, the '*Board*' and '*Sail*' operators for the Ferry domain (see also Figure 1). The left- and right-hand sides of each schema represent, respectively, preconditions and (declarative) effects of the actions. Notice that *all* places, edges and nodes appearing in the preconditions are in a *bijective* mapping with those in the effects. Thicker, dashed arrows in the preconditions are procedural representations indicating which setNode(s) should be moved and where. Constraints on the types and properties of setNodes can be specified in the the preconditions using *parameters.* For example, in the *Board* action schema, object 'y' must be an instance of type '`car`', and place 'A' must be empty. In addition, notice the use of type names (e.g., '`port`') to require, where appropriate, the presence of specific types of places.

An action schema can be applied in the current state (setGraph) S when both of the following conditions are met:

(i) all places, objects and edges present in the preconditions *match* (i.e., can be bound to) *distinct* elements of S which are arranged in identical topological structures (same 'pattern' of places, edges and objects);

(ii) for the chosen bindings, all the specified constraints are satisfied, and each of the setNodes being moved matches the type of setNodes that its destination place is allowed to contain.

The definition of '*match*' is as follows: two types match *iff* one is a sub-type of the other; two instances match *iff* they have the same label; an instance x and a type T match *iff* x is an instance of T (or of any of T's sub-types). A place (specified in an action preconditions) containing n setNodes matches any place (of the appropriate type) containing *at least* n setNodes (of the appropriate type). Matching edges must connect setNodes that match.

The *goal* of a problem will be specified simply as a (possibly typed) setGraph, allowed to contain both specific instances and typed elements, but no constraints. Given a setGraph S, the conditions for the achievement of a goal G are analogous to the requirements described in (i) for action application.

For example, it is easy to see that all elements in the preconditions of the *Sail* operator can be mapped to corresponding (distinct) elements of the setGraph shown in Figure 1, as all elements in the preconditions match corresponding (distinct) elements of the state. Note that the second part of condition (ii) is required because the destination place of an action precondition could be bound to an instance (in the state) which has *restrictions* on the type of its contents. The following example illustrates this situation.

Example 4.1 One way to represent the ToH domain using setNodes consists of considering every disk as a place allowed to contain up to one disk (place) of a smaller size. The 'peg' type can be represented as a place able to contain up to one disk of *any* size. The smallest disk can be represented as a place which is not allowed to contain any type of places. In the case of three disks, these constraints can be imposed using the following hierarchy of place types:

$$\texttt{Small}\{\ \} \dashv \texttt{Medium}\{\texttt{Small}\} \dashv \texttt{Large}\{\texttt{Medium}\} \dashv \texttt{Peg}\{\texttt{Large}\} \dashv \texttt{PLACE} \quad (4.1)$$

where '$x \dashv y$' means that type x is subtype of y, and '$type_1\ \{type_2\}$' means that places of type '$type_1$' can only contain instances (of places) matching '$type_2$'.[6] The type 'PLACE' is the root of the place-type hierarchy.

Using (4.1) as a basis, it is possible to describe the legal movements of the disks using only one operator. Figure 3.(a) depicts the initial state for a ToH problem with three disks (places) labelled 'S', 'M' and 'L' and three pegs ('P1', 'P2', 'P3'). 'S' is an instance of the type 'Small', 'M' of 'Medium', and 'L' of 'Large'. The three peg-places are instances of 'Peg'. Figure 3.(b) represents the *Move* action schema.

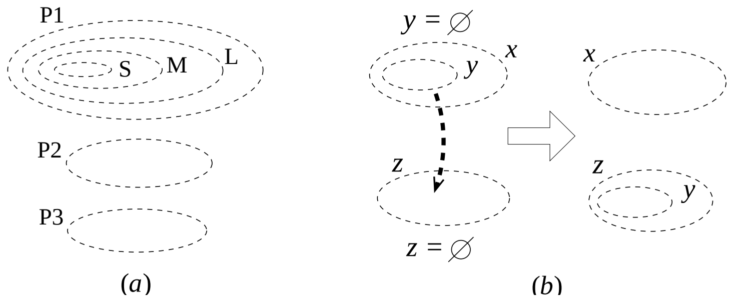

Figure 3: Tower of Hanoi domain: (a) initial state; (b) *Move* operator

To see why condition (ii) is needed, consider the preconditions of the *Move* operator. The type of the places x, y and z, unspecified, is assumed by default to be the most general (i.e., 'PLACE'). Hence, disk y and its destination place z (a disk or a peg) can be bound to (match) *any* pair of distinct places which

[6]Note that an instance of a type matches all of its *super*-types. Hence, for example, a 'Small' disk can be contained also in 'Large' and 'Peg' places.

are currently empty (clear). However, in order to prevent a larger disk from being moved into a smaller one (and avoid writing a distinct operator for each possible legal move), it is necessary to introduce the second part of condition (*ii*), which brings into the picture the constraints (imposed by formula 4.1) on which types of disks can be contained in which types of places. For example, according to (4.1), disk 'L', an instance of '`Large`' place, does not match the *type* of places allowed in 'M' or in 'S'.

4.2 Movement and State Transformation

In any physical domain containing move and change elements, an object can be characterised by its *spatial location* and by a set of attribute-value associations that determine its current *state* (e.g., type, color, temperature, size, etc.). Any of such properties, in general, can be affected by an action. Hence, the proposed paradigm should be able to represent them and to encode their possible transformations. It is immediate to see how the abstract structure of setGraph and the action schemata can be used to represent and plan the *movement* of objects (i.e., the change of spatial location).[7] If the number of possible states that an object can be in is finite, then also *state* changes can be easily represented using setGraphs. In fact, given an object x with k possible states $\{s_1, \ldots, s_k\}$, it is possible to build a setGraph having k places $\{p_1, \ldots, p_k\}$ distinct from the other existing setNodes. A new setNode x', situated in place p_j and linked to x, will encode the fact that object x is currently in state s_j. A state change $s_j \rightarrow s_i$ will be represented by *moving* x' from place p_j to place p_i. Legal movements can be specified by appropriate action schemata, according to the domain-specific constraints.

The following Lemma and Proposition present more formal properties of the expressiveness of a setGraph representation:

Lemma 4.2 Given a finite domain U with k objects $U = \{o_1, \ldots, o_k\}$, any n-ary relation $P \subseteq U^n$ having cardinality l can be represented as a setGraph containing $(k + l)$ setNodes and nl edges.

Proof Every object $o_j \in U$ can be represented by a distinct setNode s_j. Each of the 'l' relational instances $(x_1, \ldots, x_n) \in P$ can be represented as an additional setNode s_{k+i} with n edges linking it to the setNodes that correspond to the objects $x_1, \ldots, x_n$. Hence, the lemma follows immediately.

Proposition 4.2 If *non-conservative* changes are permitted, setGraph domain descriptions are *at least* as expressive as STRIPS domain descriptions.

Proof In order to prove the thesis, it will be sufficient to show that any state and operations that can be specified in STRIPS can also be encoded using the setGraph formalism. In STRIPS, a state is a collection of relational instances. From lemma 4.2, it follows that any STRIPS state can be encoded as a set-

[7]Indeed, the next section shows that this leads to more efficient planning.

Graph. In addition, each (ground) STRIPS operator consists of a 'Precondition' list, 'Add' list and 'Delete' list, each containing a conjunction of propositions (i.e., relational instances). Again, from lemma 4.2 it follows that any Precondition list can be encoded as a setGraph and used as preconditions in the left-hand side of a setGraph action-schema definition. Finally, in order to encode the addition and deletion of relational instances to/from the state as specified by the Add and Delete lists, it is sufficient to extend the setGraphs with a set of primitive, non-conservative transformations (as discussed in the previous section) that allow edges and setNodes to be dynamically added to and removed from a setGraph. Q.E.D.

5 Experimental Results

In order to assess the capability of the proposed representation to encode physical domains effectively and efficiently, various domains (including Logistics, BW, Briefcase, Rocket, ToH, Eight-puzzle, Miconic, Gripper and Robot, taken from the set of benchmarks used in the International Planning Competitions and planning literature) were translated into setGraph based representations. A prototype planner was developed, which implements a '*Cartesian*' restriction of the general representation proposed. In particular, the specific instance implemented does not allow the use of edges, but permits places and objects to be assigned identical labels to encode place-object 'associations'. In addition, places are allowed to have an internal structure consisting of a matrix (or vector) of adjacent sub-places. In other words, a place can be either a *set* of objects, or a one- or two-dimensional *array* of 'cells', each one allowed to contain up to *one string* (representing an object). In spite of the restricted expressiveness, this simpler, 'diagrammatic' version still contains most of the fundamental characteristics of the general model proposed, and allows a first assessment of the validity of its basic premises without requiring unnecessary (and possibly futile) implementation effort.

A subset of the domains listed above (namely, Eight-puzzle, Miconic, BW, Briefcase and Gripper) was re-formulated for this representation, and used to carry out an initial set of experiments. The BW encoding used in the experiments and a possible problem instance are reported in Figure 4. In order to compare the performance of the new planning representation with respect to that of a standard propositional languages, two planners were actually built, identical in all aspects except for the *representation* which they adopted. The first planner (named 'PMP', Pattern-Matching Planner) describes and solves problems using the representation just introduced, while the second planner (called 'PP', Propositional Planner) relies on the classical propositional STRIPS formalism (with types). Both planners were implemented using the same programming language (Java), adopted exactly the same search method (blind, breadth-first, forward search), and were run on the same machine. In addition, each of the five domains used for the tests was described so as to present exactly the same *search space* in each of the two representations. The

```
(define (domain Blocksworld)
  (:ObjectTypes block table)
  (:PlaceTypes stack [object::1])
  (:action PutOn
     :parameters (x - block y - object)
     :pre (stack {x | _ } stack {y | _ })
     :post (stack {_ | _ } stack {y | x })
  )
)

(define (problem Sussman)
  (:domain Blocksworld)
  (:Objects A B C - block T - table)
  (:Places s1 s2 s3 - stack)
  (:init s1 [T | A | C | _ | _ ]
         s2 [T | B | _ | _ | _ ]
         s3 [T | _ | _ | _ | _ ] )
  (:goal stack {C | B | A} )
)
```

Figure 4: Blocksworld domain description and 'Sussman' problem instance

results obtained showed a superior performance of PMP on *all* of the domains, and on *all* of the problem instances. The speed-up factor varied from a minimum of two to as much as *ninety* times faster. For reasons of space, we report in Table 1 only the values obtained with the Gripper domain (the complete set of results and the details of the domain-description language adopted are discussed in [3]). The speed-up demonstrated is a result of the ability of the representation to (*i*) *implicitly* capture the basic, common-sense physical and topological constraints of the domain, and hence perform efficient *updates* of the current state (avoiding the ramification problem), and to (*ii*) *organize* in appropriate data structures the relevant entities of the domain, allowing more efficient look-up operations.

Table 1: Time (s.) for Gripper problems ('m-n' $\equiv$ m balls in room 'A' and n balls in room 'B').

	1-0	2-0	2-1	3-0	2-2	3-1	4-0	4-1
PP	0.0	0.3	5.1	31.4	166.8	862.1	1866.4	> 16 *hours*
PMP	0.0	0.0	0.4	1.1	2.8	12.8	26.2	354.3

6 Related Work

An object-centred domain-description language (OCL_h) has been recently proposed by Liu and McCluskey in [8], based on the idea of modelling a domain as a set of objects subject to various constraints. However, OCL_h is fully sentential; consequently, some of the constraints which would be *implicit* in a setGraph still have to be explicitly declared in OCL_h (e.g., the fact that if an agent holds an object of a certain (different) type, the location of the object must coincide with that of the agent).

The work of Long and Fox [9] on the structure of planning domains and Generic Types of objects is also relevant in this context. Long and Fox have developed domain analysis techniques which allow the *automatic* detection of 'generic types' of objects (such as mobiles and 'portables') in a domain, given its purely propositional description. In essence, the physical domains considered here can be seen as combining the three abstract classes of 'transportation', 'transformation' and 'construction' domains identified by Long and Fox.

Another relevant approach is that of Cesta and Oddi [1], in which planning is seen as the problem of deciding the behavior of a dynamic domain described as a set of state variables subject to constraints. The (propositional) domain-description language proposed (DDL.1) is related to the approach taken in this paper, as it allows the description of a semantic model representing the domain as a set of possible *state changes*.

An early example of integration of model-based and sentential representations is the hybrid system of Myers and Konolige [10], who proposed a formal framework for combining analogical and deductive reasoning. In their system, operators invoked by inference rules of a sentential module (based on a logical first-order predicate language) manipulated analogical ('diagrammatic') structures which represented the state of the world. This and other related approaches falling within the area of diagrammatic reasoning [6] are particularly pertinent to the specific implementation of setGraphs used in the experiments. However, none of them appears to formulate an abstract model for encoding and solving domains within a planning framework, or to compare experimentally the *computational* efficiency of analogical and propositional representations on a given set of (planning) problems, as it has been done in the present work.

7 Discussion

The main contribution of this paper consists of having introduced a new representation tool (the setGraph) and proposed a new, model-based planning paradigm based on it, in which many of the constraints of the real-world are implicitly, effectively and naturally encoded in the domain description.

The main advantages of a setGraph representation with respect to a sentential planning formalism are that: (1) it is more efficient; (2) it is simpler, more abstract and intuitive, and (3) it better supports common-sense reasoning about action. In addition, despite being model based, the formalism is at least as expressive as any STRIPS-like propositional language (assuming that non-conservative actions are allowed). Furthermore, many 'abstract' (i.e., non-physical) domains are *isomorphic* to physical problems, and can be represented and solved using setGraphs. In particular, Fox and Long's domain analyisis techniques [9] can be used for *automatically* detecting the inherent structure of a domain from its purely propositional description, and to reveal the presence of generic types of objects even when the domain engineer is unaware of them.

In addition to offering better performance in physical domains, the representation is simpler, less error-prone and of easier use for the domain engineer. Various forms of common-sense reasoning are also better supported. For example, setGraphs facilitate heuristic-extraction, abstraction, and learning. Consider, for example, the process of macro-operator learning. The high number of possible combinations that a purely propositional, unstructured encoding involves makes this process complex and inefficient. However, sets of places and edges can be very efficiently compared and 'matched', in search of the occurrence of the same topological structure ('pattern').

One of the main limitations of the formalism is that representing state-change requires the number of possible states of an object to be *finite*. This prevents the use of numerical attributes which vary on infinite domains. In order to overcome this limitation, the representation should be extended so that objects (setNodes) can be associated to *numerical* values. This would allow the representation of mathematical functions and of other aspects of the physical world, such as time and continuous resources. Finally, the representation of uncertainty also needs to be addressed. A possible approach to this issue might be to adopt Myers and Konolige's idea of *sets* of analogical models for representing uncertainty and incomplete information [10].

In summary, the advantages of the proposed representation and its potential benefits in terms of performance motivate further investigation on setGraph and model-based planning formalisms, and on their use in *conjunction* with propositional and numerical descriptions. This paper lays the foundations for future developments in this direction.

References

[1] A. Cesta and A. Oddi. DDL.1: A formal description of a constraint representation language for physical domains. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 341–352. IOS Press (Amsterdam), 1996. (Proceedings of the 3rd European Workshop on Planning (EWSP95), Assisi, Italy, September 1995).

[2] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research – Special issue on the 3rd International Planning Competition*, 2003. (forthcoming).

[3] M. Garagnani and Y. Ding. Model-based planning for object-rearrangement problems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-03) - Workshop on PDDL*, pages 49–58, 2003.

[4] J.R. Hayes and H.A. Simon. Psychological differences among problem isomorphs. In N.J. Castellan, D.B. Pisoni, and G.R. Potts, editors, *Cognitive theory.* Erlbaum, 1977.

[5] K. Kotovsky, J.R. Hayes, and H.A. Simon. Why Are Some Problems Hard? Evidence from Tower of Hanoi. *Cognitive Psychology*, 17:248–294, 1985.

[6] Z. Kulpa. Diagrammatic representation and reasoning. *Machine GRAPHICS & VISION*, 3(1/2):77–103, 1994.

[7] J.H. Larkin and H.A. Simon. Why a diagram is (sometimes) worth ten thousands words. *Cognitive Science*, 11:65–99, 1987.

[8] D. Liu and T.L. McCluskey. The OCL Language Manual, Version 1.2. Technical report, Department of Computing and Mathematical Sciences, University of Huddersfield (UK), 2000.

[9] D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In S. Chien, S. Kambhampati, and C.A. Knoblock, editors, *Proceedings of the 5th International Conference on AI Planning and Scheduling Systems (AIPS'00)*, pages 196–205, Breckenridge, CO, April 2000. AAAI Press.

[10] K. Myers and K. Konolige. Reasoning with analogical representations. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*, pages 189–200. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1992.

[11] E.P.D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4(4):356–372, 1988.

Appendix A - Monster Change Problem (from [5])

In the Monster Change problem, three extra-terrestrial monsters are holding three crystal globes. Both monsters and globes come in exactly three sizes with no other permitted: small, medium and large. The small monster holds the medium-size globe; the medium-size monster holds the large globe; and the large monster holds the small globe. Each monster can change the globe that it is holding by shrinking it or expanding it, subject to the following constraints: (1) only one globe may be changed at a time; (2) if two globes have the same size, only the globe held by the larger monster may be changed; and (3) a globe may not be changed to the same size as the globe of a larger monster. The problem is to shrink and expand the globes so that each monster has a globe proportionate to its own size.