An Investigation into Constraint Modeling of Realistic Planning Domains

Debdeep Banerjee, Ralph Becket

National ICT Australia
debdeep.banerjee@rsise.anu.edu.au
rafe@csse.unimelb.edu.au

Abstract

Constraint Programming provides a natural way to encode combinatorial search problems. AI Planning problems can also be encoded as constraint-based search procedures. We propose an encoding of planning problems as constraint models and infer actions from state transitions. This approach provides a simple, compact encoding of the planning problems. We illustrate the encoding in a manufacturing planning domain.

Introduction

Constraint Programming (CP) is the programming paradigm that is said to be closest to the programming holy grail, i.e. the user describes the problem (models = variables + variables' domains + constraints) and the computer solves it (solving = search + constraint propagation). In constraint programming problems are formulated as constraint satisfaction problems (CSPs). It captures the problem domain's features in a natural and intuitive way. CP approaches have been used in solving many real world combinatorial search problems. AI Planning is a problem where an initial state, desired (partial) goal state and a set of operator descriptions are given and the task is to synthesize a sequence of actions to transform the initial state into a goal state. In classical planning, operators or actions are often described in STRIPS style, using three lists: preconditions, an add list and a delete list. The preconditions state which propositions must hold before an operator can be applied; the add and delete lists describe which propositions become true and which become false after applying the operator.

Both Constraint Programming and AI Planning are search problems. In constraint programming, search attempts to find a labelling of all variables consistent with the given constraints, optionally while trying to optimise for some objective function. CP search algorithms typically separate issues of variable selection (choosing which variable to branch on next), domain reduction (how to narrow the domain of the chosen variable on a search branch), and propagation (inferring necessary domain reductions on other variables as a consequence). Propagation can dramatically reduce the search space. AI planning algorithms, on the other hand, usually take a dynamic approach to problem solving by introducing new subgoals on different search paths. One weakness of this approach is that it is much harder to perform effective propagation. In ordinary CP, all decision variables and their domains are defined as part of the problem model. In ordinary AI planning, one does not know in advance what the decision variables are (here we are using "decision variable" to denote the state of something at a given time or whether a particular action is to be carried out at a given time). A CP representation of an AI planning problem therefore has to address the issue of plan length from the outset. There are two basic solutions: iterative deepening (search for a plan of length 1, search for a plan of length 2, ...) or placing an upper bound on the plan length and filling trailing action steps with "do nothing" actions.

Searching for a predefined length of plan is known as bounded length planning. In many real world planning problems, an upper bound on plan length can often be found. Interest in constraint techniques in AI planning has grown in recent years. Using constraint-based search for AI Planning has shown good results, especially in parallel optimal planning domains. Kautz and Selman (1992; 1996) first proposed a SAT encoding of the planning problems. Using the planning graph (Blum & Furst 1995) as a basis for encoding proved to be helpful as it significantly reduces the size of the SAT representation (Kautz & Selman 1999). While SAT is restricted to Boolean variables, others have extended the encoding into more general CSP based framework (Do & Kambhampati 2001). There are other approaches as well, for instance Integer Programming (Vossen *et al.* 1999), (van den Briel & Kambhmpati 2005) and Constraint Programming (van Beek & Chen 1999). The basic idea is to convert a STRIPS style planning problem into a form suitable for constraint-based search by encoding the initial states, goal states, actions, frame axioms and mutual exclusion relations as constraints for a given bounded length plan. A detailed survey of applications of constraint in AI planning is given in (Nareyek *et al.* 2005).

In this paper we investigate how we can model planning domains in the constraint model. The constraint modeling language we have used is MiniZinc (Nethercote *et al.* 2007), a subset of a more powerful constraint modeling language Zinc (de la Banda *et al.* 2006) developed as part of the G12 project (Stuckey *et al.* 2005). Here we have mod-

eled a realistic planning problem using MiniZinc in a different way than the standard constraint based encodings. Instead of transforming the STRIPS-style action descriptions of the planning problem, we encode the planning problem based on initial state, goal state and the constraints on state transitions captured from the natural description of the domain. Actions are then inferred from the state transitions. The next section of this paper describes alternative encoding of the planning problems as constraint models. Then the process of encoding is explained in greater detail with a real world planning problem. The last two sections discuss the related work and future work respectively.

Alternative Encoding

In STRIPS-style planning, state transitions are effected by actions. For example in a transportation domain we can have a "Move" action that transports objects from one location to another. There may be multiple ways to model the actions and objects in a planning domain. Detailed representation of the action is up to the domain modeller (e.g., in some formal language like PDDL (AIPS-98 Planning Competition Committee 1998)) and good modeling is an art. So for a given problem we can have different domain models and the performance of planning algorithms may vary for different models. Generally it is not an easy task to design a planning domain from a given natural-language description of the problem. On the other hand most descriptions include constraints that describe what makes a state valid or invalid. For example in the Blocks-World domain it is easy to extract the following constraints: no two blocks can occupy the same location at the same time; a block can only be moved if it is clear; a block can only move to a clear space. Along with the initial and goal state specifications, these constraints would be enough to define any standard Blocks-World problem. We can define a high level "Move" action that moves a block from one location to the other and infer its applicability based on the state transitions. For example if a block B was on the table at time step t and at next time step it is on top of another block A, then we can infer that at time step t a "Move" action took place that moved the block B from the table to the top of A. This "Move" action could be decomposed into "Pickup" and "Putdown" actions, hence we can have different abstraction levels for an action description.

The simplicity of modeling a planning problem using the domain constraints and inferring actions from the state transitions motivates us to explore the alternative encoding for the planning problems. Expressing the STRIPS-style encoding in a constraint modeling language like MiniZinc is non-trivial and feels unnatural. Modeling a planning problem using the domain objects, relations between the objects and a set of constraints of the domain is simple and natural in MiniZinc. The constraints include the initial states of objects, the object states that satisfy the goal, and intermediate constraints that describes which state transitions are legal. Here planning can be seen as abductive plan reconstruction where we can infer a sequence of actions from state transitions. We explain the encoding next by encoding a real-world planning problem.

A Realistic Planning Domain

We have modeled the Manufacturing Plant domain, which is part of the Knowledge Engineering Competition 2007 (ICKEPS 2007), with the proposed encoding above. This domain can be seen as a variant of the Job Shop Scheduling problem. A plant manufactures products that need one or more attributes. Machines in the plant are connected to each other in some configuration. There are three kinds of machines; Inputs, Outputs and Work machines. Inputs and Outputs do not make attributes and can contain any number of products in them at any point in time. Each Work machine makes one kind of attribute and can contains at most one product at a time. The goal is to find a schedule for the products moving them from the Input machines, through the Work machines (adding the required attributes) and finishing in Output machines.

This problem is in the intersection of planning and scheduling (Ruml, Do, & Fromherz 2005). The planning component is required to find paths through the machines and determine which machines should add attributes. For each product there is a set of machines that it must to go through to acquire the required attributes, but these machines may not be connected directly to each other. In this cases products need to find paths via other machines. There are many possible paths for a product, so for each product there can be many possible plans that achieve the goals of the products. The scheduling side of the problem is to schedule each product's path considering the paths of other products in the system, trying to minimize the overall finishing time. The scheduling component needs to respect the mutual resource use of the plans. In this domain the only resource is the Work machines.

Constraint Model of Manufacturing Plant Domain

To model the Manufacturing Plant problem as a constraint-state transition system, we have made two relaxations of the real problem description. We assume all the actions take unit time and all the products are in the Input machines at the initial state.

Domain Description. Modeling a planning problem as a constrained-state transition system has three main parts: 1) domain objects, 2) constraints between domain objects and 3) constraints on the behaviour of objects. The Manufacturing Plant domain has the following domain objects: Time, Machines, Products, Attributes, and Connections between machines.

Each object is defined with its total number in the domain and the set of the instances of the object. Time defines the upper bound of the time needed to finish all products. In MiniZinc objects are represented as follows:

```
int: timesteps;
int: nMachines;
int: nProducts;
int: nAttribute;

set of int: machines = 1..nMachines;
```

```
set of int: attrs    = 1..nAttribute;
set of int: prods    = 1..nProducts;
set of int: steps    = 1..timesteps;
```

Relations of the domain are either Static Relations or Dynamic Relations. Static Relations do not change over time, they are fixed for a given problem instance. There are three static relations in the Manufacturing Plant domain: the connections between machines; the set of machines making each kind of attribute; and the attribute set required by each product. These relations are described in MiniZinc as arrays of 0, 1 binary variables.

```
%Static relations
array[machines, machines] of 0..1: connections;
array[machines,  attrs]   of 0..1: can_make;
array[prods, attrs]       of 0..1: orders;
```

Dynamic relations change over time. In the Manufacturing Plant domain we have two relations that change with time: product positions and product attributes.

```
%Dynamic relations
array[steps, prods]        of var machines: pos;
array[steps, prods, attrs] of var 0..1    : achieve;
```

There are three derived variables in this model. Input and Output machines are defined as fixed sets of machines for a given problem instance. Work machines are the machines that are neither Input or Output machines.

```
set of machines : inputs;  % input machines
set of machines : outputs; % output machines
set of machines : work_machines = machines diff
                                  (inputs union outputs);
```

Next we specify the problem constraints, namely: the Initial-state constraints; the Goal-state constraints; and State-change constraints.

In the Manufacturing Plant domain the initial state constraints are that at time step 1: all the products are in Input machines (the starting machines for products can be specified for a given problem instance); and all products have empty attribute sets.

```
constraint
  forall ( p in prods ) (
    pos[1, p]  = initpos[p] /\
    forall ( a in attrs ) (
      achieve[1, p, a] = 0
    )
  );
```

The Goal-state constraints are: all products should be in output machines; each product should have the appropriate attributes.

```
constraint
  forall ( p in prods ) (
    pos[timesteps ,p] in outputs /\
    forall ( a in attrs ) (
      exists ( t in steps where t < timesteps )(
        achieve[t, p, a] = orders[p, a]
      )
    )
  );
```

There are four State-transition constraints: products cannot lose attributes; no two products can occupy the same Work machine at the same time; products can only move between connected machines (in the model we assume every machine is trivially connected to itself); and products can only acquire attributes at machines that produce them.

```
constraint
  forall ( t in steps where t < timesteps ) (
    forall ( p in prods, a in attrs ) (
      achieve[t, p, a] <= achieve[t+1, p, a]
    ) /\
    forall ( p1, p2 in prods ) (
       ( pos[t, p1] != pos[t, p2] ) \/
       ( pos[t, p1] in (inputs union outputs))
    ) /\
    forall (p in prods )(
      connections[pos[t, p], pos[t+1, p]] = 1
    ) /\
    forall ( p in prods, a in attrs )(
      achieve[t+1, p, a]  <=
      achieve[t, p, a] + can_make[pos[t, p], a]
    )
  );
```

Actions are represented by the integers 0..3 corresponding to the “Do Nothing”, “Move”, “Make” and “Move Out” actions respectively. The plan is represented by the actions carried out on the products at each step.

```
set of int: actions  = 0..3;
int: noop     = 0;
int: move     = 1;
int: make     = 2;
int: move_out = 3;

array[steps, prods] of var actions: plan;
```

We infer the plan from the state transitions of the products: if a product aquires an attribute then a ‘make action was carried out; if a product moves then a ‘move’ (or ‘move out’) action was carried out; otherwise a product is the subject of a ‘noop’ action.

```
constraint
   forall ( t in steps where t > 1, p in prods )(
       pos[t, p] != pos[t-1, p] /\
                              pos[t, p] in work_machines
       <->  plan[t-1, p] = move
     /\
       pos[t, p] != pos[t-1, p] /\
                              pos[t, p] in outputs
```

```
      <->  plan[t-1, p] = move_out
    /\
      some ( a in attrs ) (
         achieve[t, p, a] != achieve[t-1, p, a]
      )
      <->  plan[t-1, p] = make
  );
```

The following constraint specifies that at time step 1 no action should be executed on any product.

```
constraint
    forall (p in prods) (
        plan[1, p] = noop
    );
```

Problem instance. The following describes a problem instance with 5 machines, 2 products and 3 attributes:

```
timeSteps  = 4;
nMachines  = 5;
nProducts  = 2;
nAttribute = 3;

% machine X machine (reflexive)
connections = [1, 1, 1, 1, 0,
               1, 1, 1, 0, 1,
               1, 1, 1, 1, 1,
               1, 0, 1, 1, 1,
               0, 0, 0, 0, 1];

% machine X attributes
can_make = [0, 0, 0,
            1, 0, 0,
            0, 1, 0,
            0, 0, 1,
            0, 0, 0];

% products X attributes
orders = [1, 1, 0,  % A1, A2
          0, 1, 1]; % A2, A3

% Input machine set
inputs = 1;

% Output machine set
outputs = 5;

% products at starting point [input machines]
initpos = [1, 1];
```

Figure 1 decribes the machine configuration of the above problem instance.

The classical planning solution problem is a sequence of actions. In constraint programming model the solution is the valid state transitions and the inferred plan. For this example the solution is:

```
plan =  [ 0, 0, 1, 1, 2, 2, 1, 1, 2, 2, 3, 3, 0, 0 ]
pos  =  [ 1, 1, 1, 1, 3, 4, 3, 4, 2, 3, 2, 3, 5, 5 ]
```

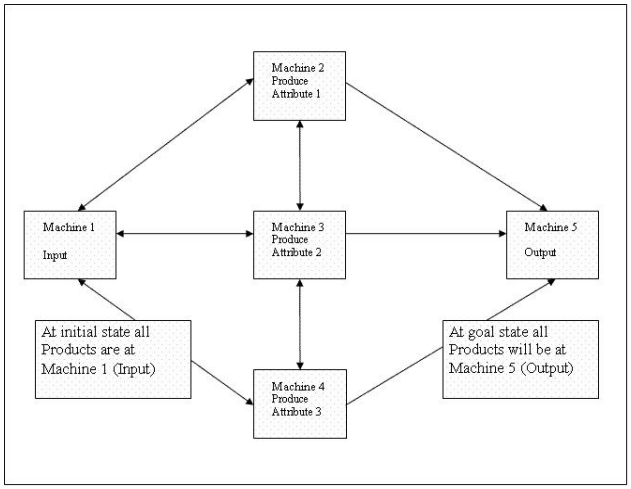

Figure 1: Machine Configuration of the problem instance

an interpretation for which is given in Table 1; note that the solution produces parallel plans for both products.

Step	Product 1	Product 2
1	Do nothing	Do nothing
2	Move from M1 to M3	Move from M1 to M4
3	Acquire attr A2	Acquire attr A3
4	Move from M3 to M2	Move from M4 to M3
5	Acquire attr A1	Acquire attr A2
6	Move out from M2 to M5	Move out from M3 to M5
1	Do nothing	Do nothing

Table 1: Action inference from State Transitions

Discussion and Related Work

In this work we have modeled a realistic planning problem in a constraint modeling language called MiniZinc. The MiniZinc model of the planning problem includes the domain objects, relations between the domain objects and the constraints of the domain. There are no STRIPS style action descriptions. The planning has been done abductively by inferring actions from the state transitions. The main three differences between the approach taken here and approaches that convert planning problem in constraint-based search problem ((Kautz, McAllester, & Selman 1996), (Do & Kambhampati 2001)) are: firstly, planning can be seen as abductive plan reconstruction as oppose to the deductive planning approach; secondly, we don't represent the actions of the domain explicitly and we dont encode the planning problem from any formal language description (for example STRIPS or PDDL). CPlan (van Beek & Chen 1999) handcodes the planning domains as CSPs, with domain specific knowledge, solves the problems using a CSP solver and extracts plans from the solutions. Though we have taken a similar approach by handcoding the planing domain in MiniZinc, we haven't encoded the actions with preconditions and effects and we infer the plan automatically from state transitions. The constrained state transition model can be seen as

an extension of the state change model for Integer Programming (IP) based planning (van den Briel & Kambhmpati 2005). The basic difference is that IP formalisms (Vossen *et al.* 1999) have defined the constraints over the state change variables using the action variables. Here the state transition constraints are extracted from the domain itself. Our plan construction process from the state transitions is similar to the planning as model checking approach. In planning as model checking (Giunchiglia & Traverso 1999) approach the semantic model of the domain is a finite state machine and transition are defined as relations with the actions. In our approach the relations are defined by the domain constraints instead of actions. Generated plan in our approach can be seen as "goal preserving plan".

Conclusion and Future work

In this paper we have described an alternative way to encode planning problems as constraint models using the MiniZinc constraint modeling language. The proposed encoding is based on the constrained-state transition system, does not represent actions as in the STRIPS style, and planning follows an abductive approach by inferring actions from state transitions. The model is based on domain objects, relations between domain objects and constraint on the transition of states. MiniZinc provides a very natural way of describing a planning problem as a constraint model. G12 platform may be an excellent platform for planning as constraint since it should seamlessly handle different kinds of problem domain. Zinc, a more expressive version of MiniZinc, will provide more flexibility for encoding planning problems as constraint systems, although it is not currently fully implemented. As a next step we want to investigate how can we guide general constraint solver with planning specific heuristics and how can we use MiniZinc to model those heuristics. In future we intend to extend our model to handle actions with different durations. Currently we can optimize the plan but optimization is solely based on minimizing the time steps. We would like to enhance our approach to handle optimum temporal planning domains with durative actions.

Acknowledgment

We want to thank Patrik Haslum for his valuable comments and suggestions. This work was supported by NICTA. NICTA is funded through the Australian governments backing Australias ability initiative, in part through the Australian research council.

References

AIPS-98 Planning Competition Committee. 1998. PDDL - The Planning Domain Definition Language. CVC TR-98-003/DCS TR-1165. Technical report, Yale Center for Computational Vision and Control.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

de la Banda, M. G.; Marriott, K.; Rafeh, R.; and Wallace, M. 2006. The modelling language zinc. In *12th International Conference on Principles and Practice of Constraint Programming*, LNCS, 700–705. Springer.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: solving the planning graph by compiling it into csp. *Artificial Intelligence.* 132(2):151–182.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *ECP*, 1–20.

ICKEPS. 2007. The international competition on knowledge engineering for planning and scheduling (ICKEPS), http://andorfer.cs.uni-dortmund.de/ edelkamp/ickeps/.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, 359–363.

Kautz, H. A., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 318–325. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kautz, H. A.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, 374–384.

Nareyek, A.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H.; Rintanen, J.; and Tate, A. 2005. Constraints and ai planning. *IEEE Intelligent Systems* 20(2):62–72.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. Minizinc: Towards a standard cp modelling language. In *13th International Conference on Principles and Practice of Constraint Programming*, LNCS. Springer. to appear.

Ruml, W.; Do, M. B.; and Fromherz, M. P. J. 2005. Online planning and scheduling for high-speed manufacturing. In Biundo, S.; Myers, K. L.; and Rajan, K., eds., *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 30–39. AAAI.

Stuckey, P. J.; de la Banda, M. J. G.; Maher, M. J.; Marriott, K.; Slaney, J. K.; Somogyi, Z.; Wallace, M.; and Walsh, T. 2005. The g12 project: Mapping solver independent models to efficient solutions. In van Beek, P., ed., *11th International Conference of Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, 13–16. Springer.

van Beek, P., and Chen, X. 1999. Cplan: A constraint programming approach to planning. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, 585–590. Menlo Park, CA, USA: American Association for Artificial Intelligence.

van den Briel, M., and Kambhmpati, S. 2005. Optiplan: A planning system that unifies integerprogramming with planning graph. *JAIR* 24:919–931.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. S. 1999. On the use of integer programming models in AI planning. In *IJCAI*, 304–309.