CPP: A Constraint Logic Programming Based Planner with Preferences

Phan Huy Tu, Tran Cao Son, and Enrico Pontelli

Computer Science Department,
New Mexico State University, Las Cruces, New Mexico, USA
{tphan|tson|epontell}@cs.nmsu.edu

Abstract. We describe the development of a constraint logic programming based system, called CPP, which is capable of generating most preferred plans with respect to a user's preference and evaluate its performance.

1 Introduction

The problem of finding a plan satisfying the preferences of a user has been discussed in [13] and recently attracted the attention of researchers in planning [2,3,5]. Indeed, PDDL, the de-facto language of the planning community, has been recently extended [9] to include constructs for the specification of users' preferences.

The original proposal described in [13] describes a model to integrate the users' preferences into a planning system based on *Answer Set Programming (ASP)*. Due to the lack of a list operator and the inflexibility in dealing with function symbols, the encoding of preferences in this system is somewhat unnatural, and it requires the introduction of artificial constant and predicates symbols. Furthermore, the encoding proposed in [13] inherits the requirement of answer set solvers that all variables have to be instantiated before computing the answer sets, possibly leading to extremely large encodings. For these reasons, the encoding in [13] is not expected to scale well to handle complex preferences.

This paper describes a *Constraint Logic Programming (CLP)* [11] based system, called CPP, for computing most preferred plans of planning problems. The detailed implementation of CPP is presented in [14]. The choice of CLP is suggested by a number of factors. First, CLP is a logic programming based paradigm, very declarative, and it is expected to allow us to maintain the underlying model proposed in [13]. Second, CLP does not require program grounding, and it allows the use of lists and other function symbols, leading to a more compact encoding of problems. CLP (and, in particular, *CLP over Finite Domains (CLP(FD))* [15]) provides the ability to express and efficiently handle arithmetic constraints, and the paradigm offers methodologies for describing search and optimization strategies. These two features appear to be vital in the context of dealing with preferences. Finally, recent studies [7] have suggested that CLP can provide an effective alternative, in terms of performance, to ASP for many classes of problems.

C. Baral, G. Brewka, and J. Schlipf (Eds.): LPNMR 2007, LNAI 4483, pp. 290–296, 2007.

The CPP system is implemented in GNU Prolog [6]—a Prolog compiler with constraint solving over finite domains capabilities. CPP accepts planning problems described in the action language $\mathcal{AL}$ [1] in terms of logic programs. Users' preferences are described in terms of logic programs as well, using the preference language $\mathcal{PP}$ from [13].

2 Background

Action Language $\mathcal{AL}$ and Planning Problems: An *action theory* of $\mathcal{AL}$ is a pair $\langle D, I\rangle$, where D is a set of propositions expressing the conditional effects of actions, the causal relationships between fluents and the executability conditions for the actions, and I is the set of propositions describing the initial state of the world. Semantically, an action theory $\langle D, I\rangle$ describes a transition diagram, whose nodes correspond to possible configurations of the world, and whose arcs are labeled with actions. A path in the transition diagram corresponds to a trajectory of a sequence of actions. We assume that domains are deterministic, i.e., there is at most one trajectory for a sequence of actions. A *planning problem* $\mathcal{P}$ in $\mathcal{AL}$ is a triple $\langle D, I, G\rangle$ where $\langle D, I\rangle$ is an action theory and G is a set of literals describing goal states. A *plan* of $\mathcal{P}$ is a sequence of actions that leads to a state satisfying G from the initial state, according to the transition diagram of $\langle D, I\rangle$.

Preference Language $\mathcal{PP}$: The language $\mathcal{PP}$ [13] supports three types of preferences: basic desires, atomic preferences, and general preferences. A *basic desire* is a temporal formula expressing desired constraints on trajectories of plans. E.g., in the transportation domain, to express the fact that a user prefers to stop by a post office, we can write

$$\mathbf{eventually}(\bigvee_{post\_office(X)} at(X))$$

or, if the user's preference is not to go by bus, we can use the formula

$$\mathbf{always}(\bigwedge_{bus\_line(X,Y)} \neg occ(bus(X,Y))).$$

In the above formulae, $occ(A)$ means that action A must occur at the current state; $\mathbf{eventually}(\varphi)$ (resp. $\mathbf{always}(\varphi)$) means that the formula φ must sometimes (resp. always) hold during the execution of the plan. An *atomic preference* provides users with a way to *rank* their basic desires, e.g., the fact that going by bus is preferred to going by subway can be expressed as $\bigwedge_{bus\_line(X,Y) \wedge subway(X,Y)} occ(bus(X,Y)) \lhd occ(subway(X,Y))$. A *general preference* is composed of atomic preferences using the connectives & (*and*), | (*or*), ! (*not*), and $\lhd$ (*preferable*). Given a preference ψ, $\mathcal{PP}$ defines an ordering relation $\prec_\psi$ between trajectories of plans: $\alpha \prec_\psi \beta$ means that the trajectory α is preferred to the trajectory β w.r.t. ψ. In this regard, a plan π is *most preferred* w.r.t. ψ if there is no plan π' s.t. the trajectory of π' is preferred to the trajectory of π.

3 System Description

We developed a system, called CPP[1], in GNU Prolog for finding most preferred plans. CPP takes as input a planning problem $\mathcal{P}$ and a set of preferences and

[1] CPP's source code is available at `http://www.cs.nmsu.edu/~tphan/software.htm`

returns as output most preferred plans of $\mathcal{P}$ w.r.t. a preference. It works by translating a planning problem into a constraint satisfaction problem, whose satisfying truth assignments correspond to plans of $\mathcal{P}$[2]. To handle preferences, CPP defines a weight function that maps plans to (integer) numbers in such a way that any plan with a maximal value of the weight function is a most preferred plan of $\mathcal{P}$. However, the other direction does not hold in general: some of the most preferred plans do not correspond to any optimal solution of the constraint satisfaction problem. If we wish to find them, we might have to use another weight function.

In our framework, a planning problem $\mathcal{P}$ is described in $\mathcal{AL}$ in terms of a Prolog program. Since GNU Prolog does not allow the symbol $\neg$, a literal $\neg F$ where F is a fluent is written as $neg(F)$. Listed below is an example of an input file for the transportation domain with three actions *walk*, *bus*, and *subway*, with obvious meaning. Initially the user is at home and his goal is to be at his office.

```
% locations, facilities & transportation media --------------------------
loc(home). loc(office). loc(stA). loc(stB). loc(stC). loc(stD).
post_office(stA). post_office(stB). phone(stB). phone(stC). phone(stD).
bus_line(stA,stB). bus_line(stC,stD). sub_line(stA,stD). sub_line(stC,stB).
pedestrian(home,stA).  pedestrian(stB,office). pedestrian(home,stC).
pedestrian(stD,office).
% actions & fluents -----------------------------------------------------
action(bus(L1,L2)):- bus_line(L1,L2). action(subway(L1,L2)):- sub_line(L1,L2).
action(walk(L1,L2)) :- pedestrian(L1,L2). fluent(at(L)) :- loc(L).
% executable condition --------------------------------------------------
executable(bus(L1,L2), [at(L1)]) :- action(bus(L1,L2)).
executable(subway(L1,L2), [at(L1)]) :- action(subway(L1,L2)).
executable(walk(L1,L2), [at(L1)]) :- action(walk(L1,L2)).
% dynamic causal laws ---------------------------------------------------
causes(bus(L1,L2), at(L2), []) :- action(bus(L1,L2)).
causes(subway(L1,L2), at(L2), []) :- action(subway(L1,L2)).
causes(walk(L1,L2), at(L2), []) :- action(walk(L1,L2)).
% static causal laws: cannot be at L1 if at L2 --------------------------
caused([at(L2)],neg(at(L1))) :  loc(L1), loc(L2), L1 \== L2.
% initial state & goal --------------------------------------------------
initially(at(home)). goal(at(office)).
```

The set of preferences is also described in terms of Prolog clauses. A basic desire is expressed as a fact of the form `basic_desire(Name,Formula)`, where `Name` is the name of the basic desire and `Formula` is a temporal formula representing the basic desire. An atomic preference is expressed as `atomic_preference(Name,Desires)`, where `Name` is the name of the atomic preference and `Desires` is a list of basic desire names. Intuitively, `Desires = [D1,..,Dk]` corresponds to the atomic preference `D1` $\lhd$ .. $\lhd$ `Dk`. A general preference is `general_preference(Name,Formula)`, where `Name` is the name of the preference and `Formula` is the formula representing the preference.

[2] Our translation is similar to SAT-based approaches to planning such as [4,12,10].

In addition to basic desires, atomic preferences and general preferences, which are supported by $\mathcal{PP}$, CPP allows another type of preferences, called *metric preferences*, which are declared as `metric_preference([(D1,W1),..,(Dk,Wk)])` where `Di`'s are basic desires and `Wi`'s are weights associated with them. The *weight* of a plan π w.r.t. a metric preference ψ is the sum of the weights of the basic desires in ψ that are satisfied by π. A plan is most preferred if it has a maximal weight value. The following is a simple example of an input file for preferences.

```
basic_desire(ds1,eventually(or(L))) :- findall(at(X),post_office(X),L).
basic_desire(ds2,eventually(or(L))) :- findall(at(X),phone(X),L).
basic_desire(ds3,always(and(L)))    :-
                findall( neg(occ(bus(X,Y))), action(bus(X,Y)),  L ).
atomic_preference(ap1,[ds1,ds2]).     atomic_preference(ap2,[ds1,ds3]).
general_preference(gp1,and(ap1,ap2)). general_preference(gp2,or(ap1,ap2)).
metric_preference(mp1,[(ds1,2),(ds2,3),(ds4,5)]).
```

In this example, the first three lines define three different basic desires. The first basic desire `ds1` describes that the user wants to be at a post office during his plan (just because he wants to send a package). The second basic desire `ds2` says that he wants to be at a phone box (to make a call, for example). The last basic desire, `ds3`, states that he does not want to go by bus.

The fourth line defines two atomic preferences: `ap1` corresponds to the atomic preference `ds1` $\lhd$ `ds2`, and `ap2` corresponds to the atomic preference `ds1` $\lhd$ `ds3`, which means that `ds1` is preferred to `ds2`, and `ds1` is preferred to `ds3`, respectively. The next line describes two general preferences: `gp1` stands for `ap1` & `ap2`, and `gp2` stands for `ap1` | `ap2`. The last line describes a metric preference consisting of basic desires `ds1`, `ds2`, and `ds3` with weights 2, 3, and 5 respectively.

The input planning problem and the preference file are compiled to CLP, and execution is initiated by issuing the predicate `main/2` to compute the most preferred plans. The first argument of `main` is the name of the preference and the second argument is the length of the plan we wish to find. For example, to find a most preferred plan of length 3 w.r.t. the basic desire `ds1` (resp. `gp1`) we can submit the query `main(ds1,3)` (resp. `main(gp1,3)`). The following is the output of the queries `main(ds1,3)` and `main(gp1,3)`:

```
| ?- main(ds1,3).
A most preferred trajectory is:
    + walk(home,stA)
    + bus(stA,stB)
    + walk(stB,office)
```

```
| ?- main(gp1,3).
A most preferred trajectory is:
     + walk(home,stC)
     + subway(stC,stB)
     + walk(stB,office)
```

4 Experiments

We compared CPP with the ASP planner in [13] on the blocks world domain (`Block`). In this domain, there are $m \times n$ blocks, numbered from $1 \ldots m \times n$. Initially, the blocks are organized in m piles, each having n blocks. The i-th pile consists of n blocks that are numbered $(1+(i-1)\times n) \ldots (i \times n)$, where the blocks

with smaller numbers are on top of the blocks with larger numbers. The goal is to have a pile of blocks from 1 to $m \times n - 1$ where blocks with larger numbers are on top of ones with smaller numbers. The location of the remaining block (block number $m \times n$), can be anywhere (i.e., on the table, on block $m \times n - 1$, or below block 1). For this domain, we tested with preferences $\psi_1, \ldots, \psi_8$ defined as follows. The first four preferences are basic desires. ψ_1 is a goal preference of having the last block (block number $m \times n$) on the top of block $m \times n - 1$ in the final state. ψ_2 is also a goal preference but it prefers to have the last block under block 1. ψ_3 and ψ_4 are state desires. ψ_3 prefers to never place any block except block 1 on the table; ψ_4 states that eventually block 1 is on block $m \times n$. ψ_5 is the atomic preference $\psi_1 \lhd \psi_2$, and ψ_6 is the atomic preference $\psi_3 \lhd \psi_4$. ψ_7 and ψ_8 are the general preferences $\psi_4 \& \psi_5$ and $\psi_8 = \psi_4 | \psi_5$.

All the experiments have been conducted on a 2.4 GHz CPU, 768MB RAM machine. Time out is set to 10 minutes. The test results are shown in Table 1. In the table, N is the length of plans that we wish to find. In each cell of the

Table 1. Performance of CPP vs ASPlan

Domain	N	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6	ψ_7	ψ_8
Block(1,4)	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		1.72	1.68	1.85	1.73	1.72	1.73	1.72	1.77
	5	0.02	0.00	0.00	0.00	0.02	0.02	0.02	0.02
		1.79	1.75	1.78	1.78	1.81	1.83	1.85	1.81
	6	0.00	0.00	0.02	0.02	0.00	0.03	0.02	0.02
		1.91	1.87	2	1.98	1.97	1.99	1.98	2.02
	7	0.02	0.00	0.02	0.03	0.00	0.03	0.03	0.03
		2.11	2.07	2.29	2.36	2.91	2.16	2.1	2.18
Block(1,5)	5	0.00	0.02	0.02	0.02	0.00	0.02	0.02	0.02
		5.65	5.51	5.63	5.73	5.53	5.65	5.73	5.74
	6	0.02	0.03	0.03	0.02	0.02	0.05	0.05	0.03
		6.85	5.72	5.76	5.97	5.99	5.89	6.11	6.18
	7	0.03	0.03	0.05	0.06	0.03	0.09	0.05	0.06
		6.15	6.27	6.19	6.13	6.18	6.32	6.53	6.45
	8	0.08	0.03	0.08	0.08	0.06	0.14	0.06	0.08
		6.53	6.51	6.69	6.56	6.56	6.71	7.14	7.28
Block(1,6)	6	0.02	0.02	0.31	0.02	0.02	0.03	0.03	0.03
		17.14	17.19	17.21	16.82	16.83	16.99	16.75	20.43
	7	0.06	0.08	0.16	0.09	0.09	0.17	0.14	0.14
		17.6	17.64	17.7	17.56	17.72	17.59	17.62	17.51
	8	0.11	0.14	0.19	0.28	0.14	0.31	0.22	0.24
		18.2	18.42	18.44	18.35	18.41	18.38	18.45	18.46
	9	0.42	0.39	0.28	0.56	0.47	0.52	0.66	0.67
		19.26	19.87	19.17	19.3	19.37	19.28	19.56	19.48
	10	2.00	0.28	0.53	0.66	2.33	0.94	2.02	2.08
		20.98	20.4	20.02	20.09	20.3	20.45	20.22	20.24
Block(1,7)	9	0.48	0.47	0.67	0.78	0.5	0.97	0.52	0.52
		32.66	32.33	33.13	32.53	32.59	32.86	32.82	32.52
	10	1.94	1.5	0.86	1.81	2.28	1.55	1.72	1.72
		34.84	34.82	34.72	34.97	34.5	34.4	38.86	35.18
	11	10.78	6.00	1.97	7.27	11.02	2.81	7.64	7.5
		38.7	38.65	38.58	38.27	38.1	38.91	38.18	38.23
	12	66.28	3.34	2.61	5.3	76.24	5.08	40.67	41.31
		42.47	42.12	42.66	42.36	42.66	45.64	42.71	43.62
Block(2,3)	5	0.2	0.23	0.36	0.38	0.24	0.55	0.56	0.66
		16.45	16.84	16.85	16.74	16.73	17.19	16.69	17.85
	6	0.12	0.18	0.2	0.47	0.11	0.44	0.22	0.2
		17.57	17.27	17.3	17.62	17.3	17.42	18.13	22.98
	7	0.25	0.31	0.23	0.75	0.27	0.38	0.34	0.33
		19.51	19.24	19.24	19.01	19.19	19.25	19.02	19.37
	8	0.82	1.15	0.28	1.92	0.81	0.89	1.05	1.05
		20.41	20.71	20.73	20.54	20.45	20.53	20.28	20.54
Block(2,4)	7	3.95	4.28	3.06	5.69	7.84	5.12	5.86	5.82
		78.97	77.21	79.95	78.77	78.76	77.63	77.75	79.04
	8	1.26	1.72	1.08	5.51	2.15	2.65	1.4	1.4
		82.53	80.87	138.38	86.54	80.8	81.05	82.08	82.56
	9	7.2	7.75	1.34	27.56	8.75	4.97	4.93	4.98
		89.53	88.68	91.09	89.17	90.09	90.68	90.26	89.21
	10	44.58	33.75	1.58	43.42	38.79	8.95	27.15	27.12
		99.02	96.81	98.24	96.64	96.08	98.7	98.46	98.34

table, the first row and the second row shows the solving times (in seconds) for CPP and ASPlan, respectively, to return a most preferred plan; TO indicates a timeout.

As can be seen from Table 1, CPP on the block world domain outperforms ASPlan on most of instances. When the number of blocks is less than or equal to 6 (problems $Block(1,4)$ and $Block(1,5)$), the solving time for CPP is negligible (less than $0.1s$), while that for ASPlan is in range from 1.7 to 7.2 seconds. However, when the number of blocks increases to more than 6 (instances $Block(1,6)$, $Block(1,7)$, $Block(2,3)$, and $Block(2,4)$), the solving time for CPP increases exponentially but is still much less than the solving time for ASPlan on most of instances.

5 Conclusion and Future Work

This paper describes a CLP based system, called CPP, for computing most preferred plans with respect to a user's preference. The preliminary results are encouraging and suggest a valid alternative for reasoning with actions and preferences. Our work is somewhat related to the work in [3] in the sense that planning problems with preferences are translated into constraint satisfaction problems. The main difference is that the work in [3] can handle preferences and constraints over *goals* only; they cannot handle preferences over *trajectories* of plans.

The system has been encoded in GNU Prolog. It is worth noting that there are other constraint programming systems, and the performance of a constraint program heavily depends on the encoding of the problem and on the underlying solver. Hence, as future work, we would like to try exploring encodings of CPP on different systems. We would also like to investigate the usefulness of heuristics and the applicability of *Constraint Handling Rules* [8] to improve the performance and extensibility of CPP. In addition, we would like to extend CPP to deal with non-deterministic and/or incomplete action theories. This involves extending the preference language $\mathcal{PP}$ so as to be able to compare plans in non-deterministic and/or incomplete action theories.

References

1. Baral, C., Gelfond, M.: Reasoning agents in dynamic domains. In Minker, J., ed.: Logic-Based Artificial Intelligence. Kluwer Academic Publishers (2000) 257–279
2. Bienvenu, M., Fritz, C., McIlraith, S.: Planning with qualitative temporal preferences. In KR, Lake District, UK (2006)
3. Brafman, R.I., Chernyavsky, Y.: Planning with goal preferences and constraints. In ICAPS, (2005) 182–191
4. Castellini, C., Giunchiglia, E., Tacchella, A.: SAT-based Planning in Complex Domains: Concurrency, Constraints and Nondeterminism. AI **147**(1-2) (2003) 85–117
5. Delgrande, J.P., Schaub, T., Tompits, H.: Domain-specific preferences for causal reasoning and planning. In KR, (2004) 673–682

6. Diaz, D., Codognet, P.: Design and implementation of the GNU prolog system. Journal of Functional and Logic Programming **2001**(6) (2001)
7. Dovier, A., Formisano, A., Pontelli, E.: A comparison of CLP(FD) and ASP solutions to NP-complete problems. In ICLP, (2005)
8. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming, Special Issue on Constraint Logic Programming **37**(1-3) (1998) 95–138
9. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Technical Report RT 2005-08-47, Dept. of Electronics for Automation, University of Brescia (2005)
10. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: preliminary report. In: Proceedings of AAAI 98. (98) 623–630
11. Jaffar, J., Maher, M.: Constraint Logic Programming. JLP **19/20** (1994)
12. Kautz, H., Selman, B.: Planning as satisfiability. In: Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92). (1992) 359–363
13. Son, T.C., Pontelli, E.: Planning with Preferences using Logic Programming. Theory and Practice of Logic Programming (2006) To Appear.
14. Tu, P.H., Son, T.C, Pontelli, E.: Planning with Preferences Using Constraint Logic Programming. Technical Report NMSU-CS-TR-2007-001, New Mexico State University (2007) http://www.cs.nmsu.edu/CSWS/techRpt/tr07-prefs.pdf.
15. Van Hentenryck, P.: Constraint Satisfaction in Logic Programming. MIT Press (1989)