CPlan: A Constraint Programming Approach to Planning

Peter van Beek and **Xinguang Chen**
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
{vanbeek,xinguang}@cs.ualberta.ca

Abstract

Constraint programming, a methodology for solving difficult combinatorial problems by representing them as constraint satisfaction problems, has shown that a general purpose search algorithm based on constraint propagation combined with an emphasis on modeling can solve large, practical scheduling problems. Given the success of constraint programming on scheduling problems and the similarity of scheduling to planning, the question arises, would a constraint programming approach work as well in planning? In this paper, we present evidence that a constraint programming approach to planning does indeed work well and has the advantage in terms of time and space efficiency over the current state-of-the-art planners.

Introduction

Constraint programming, a methodology for solving difficult combinatorial problems by representing them as constraint satisfaction problems, has shown that a general purpose search algorithm based on constraint propagation combined with an emphasis on modeling can solve large, practical scheduling problems (see, for example, (Baptiste & Le Pape 1995) and references therein). At the heart of constraint programming are constraint satisfaction problems (CSPs). A problem is represented as a set of variables, a domain of values for each variable, and a set of constraints between the variables. A solution is an instantiation of the variables that satisfies the constraints. The CSP is often solved using backtracking search and constraint programming has developed many techniques for reducing the size of the search space including adding redundant variables and redundant constraints to the CSP model .

Much work in planning, with its emphasis on a minimal domain model (just the representation of the actions) and planning specific, special purpose, search algorithms has taken an almost *opposite* approach to that of constraint programming, with its emphasis on domain knowledge and general purpose search algorithms. However, given the success of constraint programming on scheduling problems and the similarity of scheduling to planning, the question arises, would a constraint programming approach work as well in planning?

In this paper, we present evidence that a constraint programming approach to planning does indeed work well. We compare our constraint programming planner, called CPlan, to state-of-the-art planners on benchmark problems in five different domains and show that our planner has the advantage in terms of time and space efficiency and robustness. CPlan also has several other advantages which it shares with other CSP-like approaches to planning. In CPlan, the CSP model is a purely declarative representation of domain knowledge and is thus independent of any algorithm. Thus the same model can be given to a systematic solver or a solver based on local search. As well, in CPlan it is easy to represent incomplete initial states or partial information about intermediate states and to represent resource and capacity constraints. Of course, there is one important disadvantage of this approach over planners which use a minimal domain model. For each new domain, a robust CSP model must be developed. The modeling phase can require much intellectual effort and, although much of what is learned in one domain can be applied in another, each new domain does require a new model. The tradeoff is that less work needs to be done on algorithms since there are several commercial and many research constraint programming languages with general purpose constraint solvers embedded in them.

Background

We first define constraint satisfaction problems and then briefly review backtracking search (for more background on these topics see, for example, (Marriott & Stuckey 1998; Van Hentenryck 1989)).

A *constraint satisfaction problem (CSP)* consists of a set of n *variables*, $\{x_1, \ldots, x_n\}$; a domain D_i of possible *values* for each variable x_i, $1 \leq i \leq n$; and a collection of m *constraints*, $\{C_1, \ldots, C_m\}$. Each constraint C_i, $1 \leq i \leq m$, is a constraint over some set of variables called the scheme of the constraint. The size of this set is known as the *arity* of the constraint. A *solution* to a CSP is an assignment of a value $a_i \in D_i$ to x_i, $1 \leq i \leq n$, that satisfies all of the constraints.

CSPs are often solved using a backtracking algorithm. At every stage of backtracking search, there is some current partial solution which the algorithm attempts to extend to a full solution by assigning a value to an uninstantiated variable. One of the keys behind the success of constraint pro-

gramming is the idea of constraint propagation. During the backtracking search when a variable is assigned a value, the constraints are used to reduce the domains of the uninstantiated variables. The algorithm we used in our experiments, which we denote as GAC+CBJ, performs generalized arc consistency propagation and conflict-directed backjumping (Prosser 1993).

Following Van Hentenryck (1989), We say that a k-ary constraint, $k \geq 2$, is *arc consistency checkable* if at least one of its variables is uninstantiated. Such a constraint is also *forward checkable* if exactly $k-1$ of its variables have been instantiated and the remaining variable is uninstantiated. During backtracking search, the assignment of a value to a variable x_c causes some (possibly empty) set of constraints to be queued for propagation: all of the constraints that are arc consistency checkable and for which x_c is in the scheme of that constraint. For each forward checkable constraint on the queue, GAC+CBJ checks whether each value in the domain of the unassigned variable together with the values of the assigned variables satisfies the constraint, pruning those values that are inconsistent. If this process causes the unassigned variable to have all of its domain values pruned, GAC+CBJ backtracks. The arc consistency checkable constraints are processed in a similar manner: for each uninstantiated variable in the constraint, GAC+CBJ tests whether there exists values for the other variables that are consistent with the constraint, pruning those values for which this test fails and backtracking should a variable have all of its values pruned. If a variable has had its domain reduced, all of the constraints that have that variable in their scheme are added to the queue of constraints to be propagated. To backup, GAC+CBJ does not necessarily return to the chronologically most recent decision and undue that decision. Rather, it attempts to locate the source of the deadend and to jump back to that point.

Example 1 Consider the CSP with three variables x, y, and z, each with domain $\{1, 2, 3, 4\}$, and the following three constraints,

C_1: $(y \leq 3) \Rightarrow (x \geq 3)$,
C_2: $y + z \leq 6$,
C_3: alldifferent(x, y, z)

where constraint C_3 enforces that its three arguments are pair-wise different. When backtracking search starts all constraints are arc consistency checkable, but no values are pruned from the domains. Suppose backtracking search makes the assignment $x = 1$. Constraint C_1 and C_3 are queued for processing because they involve the newly instantiated variable. Processing C_1 causes the domain of y to be reduced to $\{4\}$. This causes C_2 to be added to the queue. Processing C_3 next reduces the domain of z to $\{2, 3\}$. Processing C_2 further reduces the domain of z to $\{2\}$. The rest of the search then proceeds in a backtrack-free manner.

Constraint Programming Methodology

In the constraint programming methodology we cast the problem as a CSP in terms of variables, values, and constraints. The choice of variables defines the search space and the choice of constraints defines how the search space can be reduced so that it can be effectively searched using backtracking search. We illustrate the approach using the logistics domain. In the logistics domain, there are packages which need to be moved around between cities and between locations within cities using trucks and planes.

We model each state by a collection of variables and the constraints enforce valid transitions between states. For example, in the logistics world we have the following variables for each state S_t: $C_{i,t}$, $T_{j,t}$, and $P_{k,t}$, where i, j, k range over the number of packages, trucks, and planes, respectively and t ranges over the number of steps in the plan. The domains of the package variables are locations, trucks, and planes. Assigning a package variable a location means the package is at that location in that state and assigning a package variable a truck means the package is in that truck in that state. The common STRIPS representation of this domain has two predicates that specify whether a package is at a location or in a plane or truck, respectively and an implicit state constraint that a package is either at a location or in a vehicle, but not both. This shows how CSP variables can be more succinct than propositional variables and how some state constraints can be implicitly handled. Similarly, the domains of trucks and planes are locations.

Part of the modeling task is to specify which are the visible variables and which are the hidden variables. In the logistics domain the package variables are visible and the truck and plane variables are hidden. Thus, backtracking occurs over the package variables and once they are all instantiated, the search is guaranteed to proceed in a backtrack-free manner to find values for the hidden variables.

We now turn to specifying the constraints. Constraints are represented intensionally as functions which return true or false, given a set of assignments to the variables in the scheme of the constraint. This is a compact representation, in contrast to an extensional approach where all of the assignments of values to variables which satisfy a constraint are explicitly listed (as in the planning as satisfiability framework of Kautz and Selman).

We found the following constraint categories to be useful across the five domains to which we have applied the approach. For a minimal correct model of the domain we need the action constraints which enforce how variables can change from a state S_t to a next state S_{t+1} and the state constraints which enforce how variables within a state must be consistent. The remaining categories of constraints were found to be essential in improving the efficiency of the search for a plan. Each constraint can be classified as to whether it is redundant or non-redundant. A constraint is redundant if its removal from the CSP does not change the set of solutions. Our goal is a sound and complete planner. Thus, for each non-redundant constraint that we add, we need to provide an argument that, if the set of solutions was non-empty before the addition of the constraint, it remains non-empty after its addition. In other words, a constraint must be optimality preserving to be considered for addition to the model.

Action constraints model the effects of actions. These constraints are patterned after explanation closure axioms (Schubert 1994). For example, a package variable can only

change from being at a location in S_t to being in a truck or plane in S_{t+1} (or vice-versa) and if it does change, this implies that the truck or plane must be at the same location as the package in these states.

State constraints enforce how variables within a state must be consistent.

Distance constraints are upper and lower bound constraints on how many steps are needed for a variable to change from one value to another. For example, a lower bound on the number of steps to get a package from a non-airport location in one city to a non-airport in another city is nine steps (as it needs to be loaded and unload from two trucks and one plane). These constraints were found to be among the most important for reducing the search space in the domains that we explored.

Symmetric values constraints are constraints which break symmetries on the values that variables can be assigned. For example, in the logistics domain, given two package variables, the planes in their domains are often symmetric and if there is a solution (or no solution) with a particular assignment of planes to packages, there is another solution (or no solution) with the planes swapped. With distance constraints, these constraints were found to be the most important for reducing the search space in the domains that we explored.

Action choice constraints enforce constraints on which actions can be performed in each state. Part of the explosion in the search space in planning is because a sequence of actions starting from some state can be permuted and still result in the same end state. For example, in the logistics world suppose there are two packages at an airport. A plane can either pick up both at once, or pick up one now and another later. All of these will end up being equivalent and a constraint is added which forbids all but one of the action sequences.

Capacity constraints enforce bounds on resources. In the logistics domain the trucks and planes have unlimited capacity, so these did not apply. However, in the mystery and Mprime domains (see the next section), the vehicles have capacity restrictions and there are limits on the amount of fuel available. These kinds of constraints are straightforward in the CSP approach, but difficult for traditional planners.

Domain constraints enforce restrictions on the original domains of the variables. For example, in the logistics domain, a package which is to be picked up and delivered within the same city can have its domain restricted to locations and trucks within that city.

Part of the modeling task is to specify what kind of propagation is desired for each constraint: whether a constraint should just be forward checked or arc consistency checked. Constraints of high arity are expensive to arc consistency check and may not reduce the search space enough to compensate. Experimentation is required to know whether a constraint is effective and what is the most efficient way to propagate it.

To solve an instance of a planning problem with particular initial and goal states, we start with some lower bound on the length of an optimal plan, generate a CSP model with that many steps in it, appropriately instantiate the variables in the initial and goal state, and pass the model to the backtracking algorithm GAC+CBJ. This is repeated, each time incrementing the number of steps in the plan, until a solution is found or some upper bound on the length of an optimal plan is exceeded. The idea of incrementally finding an optimal plan is due to Kautz and Selman (1992).

GAC+CBJ uses a dynamic variable ordering that selects as the next variable to instantiate the variable with the smallest domain, breaking ties by the number of constraints that the variable participates in. Thus, planning can proceed in a forwards or backwards or middle out direction and *any* part of the plan can be worked on before other parts. The overall planning algorithm, CPlan, is sound, complete, and guaranteed to terminate (but, as with other planners, the algorithm is incomplete in any practical sense since it can run for a very long time).

Experiments

We have applied our constraint satisfaction methodology to the five test domains used in the First AI Planning Systems Competition, held in Pittsburgh, June 6–9, 1998, and compared our results to four other planners: Blackbox, HSP, IPP, and TLPlan. Blackbox, HSP, and IPP were all entered into the AIPS'98 competition and each was the best or among the best in at least one of the test domains.

Blackbox (Kautz & Selman 1998a) is based on converting planning graphs (as constructed by Graphplan (Blum & Furst 1997)) into a CNF formula, and then attempting to solve the formula using a variety of satisfiability solvers. HSP (Bonet & Geffner 1998) is a forward-chaining planner which uses hill-climbing search with an automatically generated (inadmissible) heuristic cost function to estimate the distance to the goal state. IPP (Koehler & Nebel 1998) is based on Graphplan, and like Graphplan constructs a planning graph in a forwards direction and then searches it in a backwards direction to extract a plan. IPP improves on Graphplan by having a better memoization scheme to recognize subsets of goals that have failed in the past and a richer representation language. TLPlan (Bacchus 1998) is a forward-chaining planner which allows various heuristic search algorithms to be selected and provides a temporal logic for representing declarative search control knowledge. With respect to the AIPS'98 competition benchmark problems, TLPlan only comes with domain knowledge specified for the logistics problems and so we only compared its performance to the other planners on this domains.

We used the following experimental setup. All experiments were run on 400 MHz Pentium II's with 256 Megabytes of memory. Each planner was given one hour of CPU time and 256 Megabytes of memory in which to solve a problem. If the planner solved the problem within the resource limits, the CPU time was recorded (see Tables 1–5). By solving a problem, we mean that, if a plan exists, the planner returns a plan (either optimal or non-optimal), and if a plan does not exist, the planner correctly reports this fact. For some of the planning problems in the mystery domain (see Table 3), no plan exists and, by definition, a non-systematic planner such as HSP cannot correctly solve these

Table 1: Time (seconds) to solve gripper planning problems. The absence of an entry indicates that the problem was not solved correctly within the given resource limits.

	CPlan	Blackbox	HSP	IPP
1	0.01	0.11	0.03	0.02
2	0.04	5.68	0.10	0.39
3	0.08	.	0.11	7.83
4	0.17	.	0.18	100.37
5	0.28	.	0.26	.
6	0.48	.	0.35	.
7	0.75	.	0.46	.
8	1.15	.	0.53	.
9	1.67	.	0.74	.
10	2.34	.	0.94	.
11	3.17	.	1.13	.
12	4.23	.	1.45	.
13	5.52	.	1.49	.
14	7.07	.	1.81	.
15	8.92	.	2.19	.
16	11.15	.	2.56	.
17	13.67	.	3.04	.
18	16.81	.	3.26	.
19	20.19	.	3.77	.
20	24.35	.	4.23	.

problems. As well, in the Mprime domain Blackbox sometimes incorrectly reported that no plan exists when CPlan and IPP were able to find a correct plan.

We were not able to exactly duplicate the results that the individual planners obtained in the AIPS'98 competition. To varying degrees the planners require parameter tuning on each domain they are applied to. For IPP, only the default parameters were used. For Blackbox the only parameter we needed to vary from its default setting (in order to approximately equal the performance of the planner in the AIPS'98 competition in terms of number of problems solved) was to increase the respective parameter for the maximum number of nodes at each level during the planning graph generation. For the HSP planner, more elaborate parameter tuning was required.

CPlan is guaranteed to generate optimal parallel plans. Blackbox, IPP, and TLPlan can be used as either optimal or approximate planners, whereas HSP is inherently an approximate planner. In the AIPS'98 competition, Blackbox and IPP were used as approximate planners and we did the same in our experiments. For TLPlan, we used the default settings, including using depth-first search as in (Bacchus & Kabanza 1998). Thus, TLPlan was used as an approximate planner. We found in our experiments that Blackbox and TLPlan generated high quality plans that were almost always optimal or nearly optimal. However, HSP often generated longer plans. For example, in the gripper domain, the length of the plans generated by HSP almost doubles the length of the optimal plan for each instance. For HSP, sometimes the quality of the plans can be improved with different parameter settings, but with the consequence that many fewer instances are solved.

Blackbox and IPP consume large amounts of memory and often ran out of this resource before finding a plan. Is it just a matter then of more memory and these methods could solve the problems? To examine this question, we ran the following experiments. Our machines have 256 Mb of physical memory, but processes are permitted to allocate up to 640 Mb (this is a preset limit in our configuration of Linux). The AIPS'98 competition consisted of two rounds and within a round the problems are roughly ordered by difficulty. with the easier problems coming first. For instance 3 of the logistics problems (third row of Table 2), CPlan is able to find a plan in under one second of CPU time using just under 2 Mb of memory. Blackbox on this problem exhausts the available 640 Mb quickly (in about three minutes) without finding a plan. IPP exhausts the available 640 Mb more slowly (in just over eleven hours) but also without finding a plan. For instance 6 of the mystery problems, CPlan is able to find a plan in just over three seconds of CPU time using under 2 Mb of memory. Blackbox on this problem takes 83 seconds and uses 568 Mb to find a plan. IPP on this problem takes 133 seconds and 107 Mb of memory to find a plan. Thus, on these smaller problems, CPlan can be one to two orders

Table 2: Time (seconds) to solve logistics planning problems. The absence of an entry indicates that the problem was not solved correctly within the given resource limits.

	CPlan	Blackbox	HSP	IPP	TLPlan
1	0.05	1.48	.	0.62	0.37
2	0.06	4.29	.	552.20	1.48
3	0.94	.	.	.	15.83
4	0.17	.	.	.	44.37
5	1.73	148.01	1.17	2.52	0.28
6	18.89	.	.	.	68.80
7	0.09	.	.	3059.24	4.65
8	0.16	.	.	.	78.60
9	0.32	.	.	.	176.41
10	.	.	.	.	135.44
11	0.05	4.51	4.96	6.48	4.43
12	0.12	.	.	.	231.97
13	0.59	.	.	.	865.04
14	0.68	.	.	.	651.37
15	203.15	.	.	.	19.24
16	0.40	.	.	.	136.59
17	0.32	.	.	1935.52	74.97
18	1308.15	.	.	.	3592.67
19	1.37	.	.	.	2308.24
20	28.94	.	.	.	2897.54
21	0.72	.	.	.	1684.82
22	.	.	.	.	.
23	.	.	.	.	96.37
24	0.09	.	.	.	562.35
25	13.94	.	.	.	.
26	.	.	.	.	.
27	48.62	.	.	.	.
28	.	.	.	.	.
29	.	.	.	.	.
30	.	.	.	.	.
1	0.00	0.28	0.16	0.16	0.06
2	0.01	0.36	0.24	0.18	0.13
3	0.06	0.56	1.14	0.37	0.40
4	0.07	124.03	.	.	2.44
5	0.94	.	.	49.39	1.09

Table 3: Time (seconds) to solve mystery planning problems. The absence of an entry indicates that the problem was not solved correctly within the given resource limits.

	CPlan	Blackbox	HSP	IPP
1	0.00	0.11	0.05	0.08
2	0.03	4.22	7.09	11.43
3	0.02	0.42	0.39	0.85
4	0.00	1.18	.	0.37
5	0.00	11.36	.	7.58
6	3.06	.	.	133.95
7	0.00	1.14	.	9.79
8	0.00	.	.	30.88
9	0.03	1.03	0.66	1.32
10	0.53	.	86.45	.
11	0.02	0.53	0.07	0.28
12	0.00	0.94	.	0.49
13	0.39	.	.	.
14	1.08	.	.	.
15	0.39	.	16.81	.
16	0.00	3.70	.	5.70
17	0.12	2.44	11.91	29.68
18	0.00	13.39	.	273.20
19	0.13	5.26	6.05	19.94
20	0.53	.	6.23	.
21	0.00	.	.	50.25
22	0.00	.	.	.
23	0.00	.	.	101.19
24	0.00	.	.	72.39
25	0.01	0.10	0.06	0.07
26	0.07	1.24	0.43	1.44
27	0.01	0.42	0.56	0.69
28	0.02	0.39	0.11	0.18
29	0.01	0.38	0.32	0.59
30	0.15	3.91	6.42	9.42

of magnitude more efficient in both time and space. Further, the difference between the intensional representation used by CPlan and the extensional representations used by Blackbox and IPP only grows as the problem sizes increase.

In these experiments, the planners without domain knowledge appear to be quite brittle—either solving a problem quickly or not solving it at all—and to not scale well to more difficult problems.

Related Work

In this section, we relate our work to previous work in planning. We first review previous CSP-like approaches to planning and then we review previous work on using domain specific declarative knowledge to improve planning.

The definition of a CSP is general enough that it subsumes Boolean satisfiability and integer linear programming; both of these can be viewed as particular restrictions on the domains of the variables and the forms of the constraints. The first work that we are aware of that casts planning as a CSP is the work on planning as satisfiability by Kautz and Selman (1992; 1996). Our work owes much to theirs, including the general framework of planning as satisfiability, and the idea of a state-based model with explanation closure axioms defined on state variables and no variables that explictly model

Table 4: Time (seconds) to solve Mprime planning problems. The absence of an entry indicates that the problem was not solved correctly within the given resource limits.

	CPlan	Blackbox	HSP	IPP
1	0.05	0.59	0.14	2.34
2	0.09	4.42	13.92	23.93
3	0.16	0.60	0.51	4.35
4	0.22	0.60	0.61	2.51
5	0.22	.	.	7.19
6	2.25	.	201.81	.
7	0.13	.	.	12.58
8	0.67	1.99	4.39	32.91
9	0.06	1.34	0.75	4.93
10	2.14	.	273.32	.
11	0.12	1.28	0.18	4.19
12	0.09	1.63	0.87	5.92
13	2.38	.	.	.
14	4.04	.	.	.
15	1.39	.	104.02	.
16	0.13	3.61	3.33	.
17	0.57	.	27.76	91.76
18	6.95	.	.	.
19	0.61	.	30.48	.
20	2.34	.	130.53	.
21	0.22	.	.	58.16
22	3.50	.	187.12	.
23	2.10	.	.	.
24	0.64	.	.	.
25	0.01	0.13	0.06	0.92
26	0.27	2.22	1.11	12.86
27	0.05	2.21	6.83	31.49
28	0.07	1.33	0.64	4.65
29	0.09	1.57	1.30	15.90
30	0.80	.	.	.
1	0.16	0.57	0.74	7.12
2	0.14	1.74	0.67	1.83
3	1.20	.	.	.
4	0.07	1.13	1.07	7.88
5	0.04	0.33	0.14	1.95

actions. More recently, Bockmayr and Dimopoulos (1998) have examined integer linear programming models of planning. They use 0-1 integer variables in their formulations in a manner similar to the satisfiability approach and examine the effect of adding redundant constraints. Their work is preliminary and it is as yet unclear whether the approach will be fruitful. (For a problem in the logistics world they report a time of 75 minutes on an unknown machine to find a plan; we are able to solve this problem in less than one minute.) As well, there has been a long history of partial order planners which are often referred to as performing constraint posting. In these approaches, constraint satisfaction techniques are added as an adjunct to the planning process, but the planning process itself is not formulated as a CSP.

There have been two streams of work on adding declarative domain knowledge to improve the performance of planners. In the first stream, the knowledge is hand-coded as in our approach. In the second, the knowledge is automatically derived. As two examples, Kautz and Selman (1998b) advocate adding domain specific knowledge in a declarative

Table 5: Time (seconds) to solve grid planning problems. The absence of an entry indicates that the problem was not solved correctly within the given resource limits.

	CPlan	Blackbox	HSP	IPP
1	0.67	8.08	1.13	3.03
2	33.36	.	4.33	9.38
3	.	.	.	.
4	1773.28	.	.	57.00
5	.	.	.	.

manner to a planner and show some limited experimentation in a satisfiability-based planner, and Bacchus and Kabanza (1998) provide a temporal language for specifying domain knowledge and show how effective it is in their TLPlan planner. For work on automatically deriving constraints from action representations and initial and goal states, HSP (1998) derives distance constraints; Fox and Long (1999), show how to identify a primitive form of symmetry and use it in a planner; Gerevini and Schubert (1998) show how to derive state constraints; and Nebel, Dimopoulos, and Koehler (1997) show how to ignore irrelevant facts and operators, all to automatically improve the performance of planners. This work may also be helpful in semi-automating the task of developing CSP models for planning.

Conclusions

We presented a constraint programming or constraint satisfaction approach to planning. The approach shares the advantages of other CSP-like approaches, including the expressiveness of the modeling language, the declarativeness of the models, and the independence of the model from the solving algorithm. We also demonstrated that a constraint programming approach has several distinct advantages over other approaches, including the succinctness of the models, and the robustness and speed with which plans can be found. Our experiments indicate that present state-of-the-art planners can be brittle, either solving the problem quickly or not at all. Our system, CPlan, can be one to two orders of magnitude more efficient in both time and space on problems which the other systems can solve and can scale to harder problems which the other systems cannot solve.

For future work, we intend to look at approximate planning, by examining whether the same declarative CSP models that we solved using a systematic search algorithm in the experiments presented in this paper can be solved effectively using local search algorithms. As Blackbox has shown, this can be an effective technique. As well, we intend to look at alternative CSP models. It is well known within the operations research and constraint programming fields that one of the keys to effectively solving difficult combinatorial problems is to find the right model of the problem. In this paper we presented results for a state-based model. The question remains if this is the best CSP model for planning.

Availability. CPlan, including source code and CSP models for the domains discussed in this paper, is available via http://www.cs.ualberta.ca/~vanbeek.

Acknowledgements. We would like to thank Fahiem Bacchus for suggesting that we evaluate our methodology on the planning problems used in the AIPS'98 competition. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

Bacchus, F., and Kabanza, F. 1998. Using temporal logics to express search control knowledge for planning. Unpublished manuscript.

Bacchus, F. 1998. TLPlan (Version of September 1998). http://logos.uwaterloo.ca/~fbacchus.

Baptiste, P., and Le Pape, C. 1995. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In *IJCAI-95*, 600–606.

Blum, A. L., and Furst, M. L. 1997. Fast planning through plan graph analysis. *Artif. Intell.* 90:281–300.

Bockmayr, A., and Dimopoulos, Y. 1998. Mixed integer programming models for planning problems. In *CP98 Workshop on constraint problem reformulation*.

Bonet, B., and Geffner, H. 1998. HSP (Version of August 1998). http://www.ldc.usb.ve/~hector.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning domains. Technical Report 1, Durham University, UK.

Gerevini, A., and Schubert, L. 1998. Inferring state constraints for domain-independent planning. In *AAAI-98*, 905–912.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *ECAI-92*, 359–363.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI-96*, 1194–1201.

Kautz, H., and Selman, B. 1998a. Blackbox (Version 3.1). http://www.research.att.com/~kautz.

Kautz, H., and Selman, B. 1998b. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proc. of the 4th International Conference on AI Planning Systems (AIPS-98)*.

Koehler, J., and Nebel, B. 1998. IPP (AIPS'98 version). http://www.informatik.uni-freiburg.de/~koehler

Marriott, K., and Stuckey, P. J. 1998. *Programming with Constraints*. The MIT Press.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proc. of the European Conference on Planning (ECP-97)*, 338–350. Springer Verlag.

Prosser, P. 1993. Hybrid algorithms for the constraint satisfaction problem. *Comput. Intell.* 9:268–299.

Schubert, L. 1994. Explanation closure, action closure, and the Sandewall test suite for reasoning about change. *J. of Logic and Computation* 4:679–700.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*. MIT Press.