Generalizing GraphPlan by Formulating Planning as a CSP

Adriana Lopez
Dept. of Computer Science
University of Toronto
Toronto, Ontario
Canada, M5S 3G4
alopez@cs.toronto.edu

Fahiem Bacchus
Dept, Of Computer Science
University of Toronto
Toronto, Ontario
Canada, M5S 3G4
fbacchus@cs.toronto.edu

Abstract

We examine the approach of encoding planning problems as CSPs more closely. First we present a simple CSP encoding for planning problems and then a set of transformations that can be used to eliminate variables and add new constraints to the encoding. We show that our transformations uncover additional structure in the planning problem, structure that subsumes the structure uncovered by GRAPHPLAN planning graphs. We solve the CSP encoded planning problem by using standard CSP algorithms. Empirical evidence is presented to validate the effectiveness of this approach to solving planning problems, and to show that even a prototype implementation is more effective than standard GRAPHPLAN. Our prototype is even competitive with far more optimized planning graph based implementations. We also demonstrate that this approach can be more easily lifted to more complex types of planning than can planning graphs. In particular, we show that the approach can be easily extended to planning with resources.

1 Introduction

A powerful technique for solving planning problems, first used in [Kautz and Selman, 1996], is to impose a fixed bound on plan length. This converts the problem to one that lies in NP. The resulting problem can then be solved by encoding it in any of a number of alternative NP-complete formalisms, e.g., SAT [Kautz and Selman, 1996], integer programming [Vossen *et ai%* 2001], or *constraint satisfaction problems* (CSPs) [van Beek and Chen, 1999; Do and Kambhampati, 2001], and then using the powerful algorithms that have been designed to solve these general problems.

In this paper we examine the approach of encoding planning problems as CSPs more closely. We use a very simple encoding to translate the planning problem into a CSP. The advantage of using CSPs in general, and our encoding in particular, is that it preserves much of the structure of the original planning problem, and thus various transformations that utilize this specific structure can be applied to create a new CSP that is easier to solve.

The utility of exploiting planning specific structure has been convincingly demonstrated by the BLACKBOX [Kautz and Selman, 1998] and GP-CSP [Do and Kambhampati, 2001] planners. These systems encode a transformed representation of the planning problem called the *planning graph* to SAT and CSP respectively. The planning graph is a planning specific construction, due to [Blum and Furst, 1997], that computes and represents some important additional structural information about the planning problem. Thus by encoding the planning graph rather than the original planning problem, BLACKBOX and GP-CSP are able to capture more of the structure of the planning problem in their encoding, which results in a significant improvement in performance.

In our approach we bypass the construction of a planning graph. Instead, we study techniques that can equally well exploit the structure of the planning problem and at the same time work directly on the CSP encoding. We demonstrate that it is possible to develop transformations that uncover all of the additional structure obtained from planning graphs.

The advantage of our approach is that it works directly on a much richer and more robust representation. Consequently we obtain at least three important advances over the planning graph construction: (1) Our approach allows us to *generalize planning graphs.* In particular, we can enhance our approach to extract other kinds of structure to allow us to solve the planning problem more efficiently. (2) We can more easily extend our approach to more *complex types of planning problems,* e.g., those involving resource usage. (3) Since the final result is a CSP we can *automatically utilize CSP solution techniques* which can be more powerful than standard GRAPHPLAN searching methods. In the paper we will provide evidence for all of these points.

First, we present our CSP encoding of a planning problem. Then we present a set of transformations that can be used to eliminate variables and add new constraints to the encoding, and show that these transformations subsume and generalize planning graphs. Then we present empirical evidence to validate the effectiveness of our approach to solving planning problems. To demonstrate that our approach can be more easily lifted to more complex types of planning, we show how it can be easily extended to planning with resources. Finally we close with some conclusions and a description of future work.

2 Generating the BASE-CSP

To encode a planning problem as a CSP we impose a bound k on the "size" of the plan. In this paper we will measure plan size by the number of GraphPlan-concurrent steps (GP-steps) in the plan. It will become apparent in the sequel what GP-steps means.

In k GP-steps each proposition or action can change at most k times. So to encode a k GP-step plan in a CSP we can utilize fc -f 1 sets of propositional variables P_i^s, and k sets of action variables A_j^s, where s ranges from 0 to k for propositions and 0 to k - i' for actions, and i and j range over the number of distinct propositions and action instances in the planning problem respectively. Each of these variables will be boolean. Intuitively, P_i^s means that proposition $P\{$ is true at GraphPlan step s, and A_j^s means that action instance Aj was executed at GraphPlan step ($\neg P_i^s$ and $\neg A_j^s$) t e the opposite).

Clearly by setting each of these variables we can capture any k GP-step plan and its effects. However, many illegal plans and nonsensical outcomes are also captured. We need constraints to ensure that any solution to the CSP (i.e., any setting of the variables that satisfies all of the constrains) is a legal plan. There are various possible sets of constraints that can serve our purpose. Here we present one particular set:

1. *Unary initial state and goal constraints.* The Dropositional variables from step ze P_i^0, and from :k, P_i^k are required to have values compatible with the initial and goal states of the planning problem.

2. *Precondition constraints.* An action A_j^s cannot be true unless its precondition is satisfied. This Rives the constraint $A_j^s \rightarrow \mathrm{Pre}(A_j)^s$, where $\mathrm{Pre}(A_j)^s$ is A_j's precondition relative to GP-step *s.*

3. *Successor state constraints.* Implicit in classical planning representations is the frame assumption: predicates not mentioned in the action's description are unaffected by the action. We can capture the effects of actions, including the implicit non-effects in a number of different ways. Here we use Reiter's formulation of successor state axioms [Reiter, 2001]. In particular, for each propositional variable P_i^s at GP-step s (a > 0) we have a constraint between it and the same proposition at GP-step .s-1. The constraint says that P_i^s is true if and only if some action made it true, or it was previously true and no action made it false. Thus the successor state axiom constraints take the form:

$$P_i^s \leftrightarrow \bigvee_{A_j \in Create(P_i)} A_j^{s-1} \vee (P_i^{s-1} \wedge \bigwedge_{A_j \in Delete(P_i)} \neg A_j^{s-1}),$$

where *Creatc(Pl)* and *Delete(Pt)* are the set of actions that create and delete *P{.* It is easy to automatically generate a set of successor state constraints from a set of STRIPS action descriptions. Additionally, by using successor state constraints that mention additional propositions from step .s-1, it is easy to encode A D L [Pednault, 1989] operators as a set of successor state axioms.[1]

'Note that the successor state constraints do not encode the action preconditions. These arc encoded as separate constraints. Thus, the only complication with A D L actions has to do with conditional effects. If Pi is a conditional effect of action A_j, subject to the condition ϕ, the successor state constraint will have a disjunct "$A_j^{s-1} \wedge \phi^{s-1}$". That is, P_i^s will be true if A_j was executed *and* condition ϕ held at the previous step.

4. *GP-Concurrency Constraints.* The above successor state axiom allows the unintended model in which we have two actions at the same step, one creating a proposition and one deleting it. To avoid such unintended solutions we must restrict concurrent actions in some way. The most natural way is a serial constraint, which says that only one action variable can be true at any step.[2] Another type of constraint is one that imposes GraphPlan concurrency. Basically it asserts that two actions cannot be simultaneously true if they interfere with each other. In this work we have chosen to use the GraphPlan (GP) concurrency constraints.

5. *Non-Null Steps Constraints.* We impose the constraint that for every step s at least one action variable should be true. This blocks null steps in the plan, in contrast LBlum and Furst, 1997] and [Do and Kambhampati, 2001] both allow null plan steps.

We will refer to this set of variables and constraints as the BASK-CSP. Any solution to the BASR-CSP will contain a setting of the action variables that comprises a valid GP-concurrent plan. If the BASE-CSP has no solution then no k GP-step plan exists for the problem.

3 Reduction of the BASE-CSP

Given a BASE-CSP representing a k-step planning problem we can use various transformations to modify it, generating a new CSP that is empirically easier to solve, and that is equivalent to the BASe-CSP in the sense that any solution to the new CSP can easily be extended to a solution to the original BASE-CSP. These transformations include inferring new constraints that can be added to the CSP and eliminating various single valued variables. Our transformations are related to the well known techniques of enforcing local consistency on a CSP prior to solving it [Freuder, 1985], however they are based on taking advantage of the specific logical form of the above set of constraints.

3.1 Adding GraphPlan Mutex Constraints

A well known technique for making CSPs easier to solve is to add redundant constraints, e.g., [Getoor *et ai,* 1997], which are constraints that make explicit additional relations between the variables. Redundant constraints that are useful to add are usually determined by examining the structure of the particular CSP. This is precisely what is done in planning graphs, where insights into the manner in which actions and propositions interact, are used to generate a new set of binary constraints called mutexes.

New binary constraints can be added to a CSP by enforcing *2-j consistency* [Freuder, 1985]. A CSP is *2-j consistent* if for every valid assignment to a pair of variables, there is a set of values for every j additional variables such that the $2+j$ assignments satisfy all of the constraints over these variables. Making a CSP 2-j consistent can be very expensive as it involves testing all sets of $2+j$ variables, and each test in the worst case takes time exponential in j. In planning problems there can be thousands of variables, so it would be impossible

[2]Serial actions can be used to represent true concurrency by adding timestamps to the states [Bacchus and Ady, 2001; Pirri and Reiter, 2000].

to make the entire BASE-CSP 2-j consistent even for small j. The contribution of GRAPHPLAN is that it demonstrated a technique that quickly achieves a very effective partial form of 2-j consistency over a limited collection of $2+j$ variables.

We can use the CSP representation to directly compute the binary mutex constraints generated by planning graphs. In this manner we lift the mutex computation to a more general framework where it can be more easily generalized.

Mutexes in GraphPlan are generated by three simple rules: (1) *Base Action Mutexes.* Actions in the same GP-step with interfering effects and preconditions are mutex. (2) *Additional Action Mutexes.* Two actions in the same GP-step are mutex if there is a pair of propositions, one from each action's preconditions, that are mutex at this GP-step. (3) *Propositional Mutexes.* Two propositions, P_i^s and P_j^s, at the same GP-step, are mutex if every action that achieves P_i^s is mutex with every action that achieves P_j^s. We can create all of these binary constraints by testing for very similar conditions in the CSP encoded planning problem:

Base Action Mutexes. These mutexes are already present in the BASH-CSP; they are the GP-concurrency constraints.

Additional Action Mutexes. The GRAPHPLAN condition for detecting additional action mutexes only handles actions with conjunctive preconditions, e.g., STRIPS actions. Thus if we have a pair of preconditions P_i^s for A_i^s and P_j^s for A_j^s, the corresponding precondition constraints $A_i^s \rightarrow P_i^s$ and $A_j^s \rightarrow P_j^s$, and a mutex constraint $\neg P_i^s \vee \neg P_j^s$, a simple 2-2 consistency test with these variables and constraints allows us to derive the new mutex constraint $\neg A_i^s \vee \neg A_j^s$. In fact, we can short-circuit this test by using the presence of the mutex constraint between P_i^s and P_j^s to immediately mark all actions with precondition P_i^s as being mutex with all actions with precondition P_j^s. Furthermore, if an action A_k^s contains both P_i^s and P_j^s in its precondition we obtain the constraint $\neg A_k^s \vee \neg A_k^s$, i.e., we obtain a unary constraint that forces A_k^s to be false. As we will describe below, single valued variables can be used to further reduce the CSP. A similar approach can be followed with more general types of preconditions using more general local consistency tests, e.g., with ADL actions where the GRAPHPLAN test is no longer valid.

Propositional Mutexes. For propositional mutexes we have one added complication, which is that our encoding, unlike planning graphs, does not utilize no-ops. Nevertheless, a general condition for when two propositions P_i^s and P_j^s are mutex can be given by examining the three cases.

1. $\neg P_i^{s-1} \wedge \neg P_j^{s-1}$ holds. In this case for P_i^s and P_j^s both to be true in the next state they must both be created by an action. In particular, neither proposition can remain true by inertia. For a mutex between P_i^s and P_j^s to hold we must have for every action A_i^{s-1} creating P_i^s that $\neg A_i^{s-1} \vee \bigwedge_{A_j \in Create(P_j)} (A_i^{s-1} \rightarrow \neg A_j^{s-1})$. That is, either the action is not possible, or it must be mutex with all actions that create P_j^s.[3] A similar condition must be satisfied by the actions creating P_i^s.
2. $\neg P_i^{s-1} \wedge P_j^{s-1}$ (or analogously $P_i^{s-1} \wedge \neg P_j^{s-1}$) holds. In this case for both P_i^s and P_j^s to hold P_i^s must be created by an action. For the mutex to hold we must have for every action A_i^{s-1} creating P_i^s that $\neg A_i^{s-1} \vee (A_i^{s-1} \rightarrow \neg P_j^s)$. That is, either the action is not possible or it must delete P_j^s (else P_j^s can be true by inertia).
3. $P_i^{s-1} \wedge P_j^{s-1}$ holds. In this case we must have for every action A^{s-1} that $\neg A^{s-1} \vee (A^{s-1} \rightarrow \neg P_i^{s-1}) \vee (A^{s-1} \rightarrow \neg P_j^{s-1})$. That is, either the action is not possible, or it must delete at least one of P_i or P_j. Note that this condition relies on the fact that at least one action must be executed at every step (the non-null steps constraint).

As in planning graphs, We can use the constraints added at step i to test for new mutex constraints at step $i+1$. The following result can then be easily proved (we omit the proof for reasons of brevity).

Proposition 1 *// the planning graph construction has detected a mutex constraint between two variab V_i^s and V_j^s (action or propositional variables), then after transforming the* BASH-CSP *to add the above mutex constraints we will have that eith(a) both variables in the CSP will have become single valued with not both being true or (h) there will be a binary constraint in the CSP between these variables that prohibits them from both being true.*

That is, the transformed CSP will capture all of the planning graph mutexes. Furthermore, it might contain *more mutexes* than those inferred by the planning graph. In the simplest case this can arise from the last case presented above where every action deletes one of P_i or P_j. This can create a new mutex in the middle of the plan. In planning graphs, on the other hand, once P_i and P_j appear in the graph without a mutex between them, they can never become mutex again (due to the presence of no-ops and the possibility of null steps in the plan).

3.2 Symbolic Reduction of Single Valued Variables

In any CSP $\mathcal{P}$, if $\mathcal{P}$ has a variable V with only a single value a in its domain, we can reduce $\mathcal{P}$ by making the assignment $V \leftarrow a$, to produce a new CSP $\mathcal{P}'$ that does not contain the variable V. $\mathcal{P}'$ has the property that for any solution S of $\mathcal{P}'$, $S \cup \{V \leftarrow a\}$ is a solution to the original $\mathcal{P}$. To reduce $\mathcal{P}$, we replace every constraint mentioning V, e.g., $C(V, V_1, \ldots, V_k)$ by a new constraint $C'(V_1, \ldots, V_k)$ such that $C'(V_1 \leftarrow x_1, \ldots, V_k \leftarrow x_k)$ is true of a tuple of assignments $(x_1, \ldots, x_k)$ if and only if $C(V \leftarrow a, V_1 \leftarrow x_1, \ldots, V_k \leftarrow x_k)$ is true.

Our translation from a planning problem produces a BASB-CSP in which every constraint C is represented *symbolically* by a logical formula. We can simplify this logical formula directly by replacing the single valued variable V by its value, and then performing standard logical reductions, e.g.,

[3]Note that if A_i^{s-1} deletes P_j^s then it will be mutex with all actions creating P_j^s via the GP-concurrency constraints.

FALSE$\wedge\alpha \Rightarrow$ FALSE, FALSE $\vee\,\alpha \Rightarrow \alpha$, etc. This yields a new logical formula that represents the reduced constraint C".

In the case of the BASH-CSP we also know the form of its different constraints. This allows us to precompute many types of constraint reductions, so we can realize many types of single valued variable reductions more efficiently.

The reduction of a single valued variable, or the application of some of the other transformations described below, might generate new single valued variables that can in turn be reduced. It can also interact with the generation of mutex constraints, allowing new mutexes to be detected. For example, say that we have determined that a propositional variable P_j^s must be false. If P_j^s appears in any conjunctive preconditions, e.g., $A_j^s \rightarrow P_j^s$, wc will also immediately infer $\neg A_j^s$, i.e., that A_j^s has also become single valued (false). With A_j^s false it can no longer be a candidate for producing some other proposition P_i^{s+1}, and we could possibly infer a new mutex involving P_i^{s+1} since one of the ways of making it true is now blocked. With mutexes, single valued variable reduction subsumes reachability analysis of planning graphs.

Proposition 2 *The variables that remain in the CSP after single valued variables have been reduced, are a subset of the variables appearing in the planning graph.*

4 Beyond GRAPHPLAN

Besides subsuming, and slightly generalizing, the planning graph inferences, further transformations can be utilized on the CSP. These transformations either add more constraints to the CSP or cause further variables to become single valued. In either case, like the planning graph inferences, they are used to make the CSP easier to solve. These additional constraints can also feed into the standard GraphPlan reductions described above, and thus generate even further mutexes.

4.1 Additional Binary Constraints

Graphplan mutexes only prohibit the single pair of values for two variables (TRUH,TRUE). Our representation treats negated propositions and actions in an entirely symmetric fashion. This means that the analysis for mutexes given above will allow us to compute mutexes between any two values. For example,when applied tP_i^s and $\neg P_j^s$, P_i^s ı g mutex with $\neg P_j^s$ means that P_i^s is"*mutin*" with P_j^s; **i.e.**, these variables must both be true or both be false. Such binary constraints can be detected and added to the BASE-CsP.

4.2 Single Valued Variable Reduction beyond GRAPHPLAN Reachability

As demonstrated above, single valued variable reduction is as powerful as the reachability analysis inherent in the planning graph construction. However, in the CSP encoding it is more powerful, obtaining, for example, the following additional reductions:

1. Propositions that are never modified by any action, are propagated without change in the planning graph. Since these propositions have no creators nor deletors, their successor state axiom reduces all of them to being equivalent to their status in the initial state, i.e., they will all become single valued. Thus they can be reduced from the CSP prior to searching for a solution.
2. If all possible actions at step *s* create proposition P_i^{s+1} then P_i^{s+1} is forced to be true. Similarly if it is deleted by all possible actions then it is forced to be false, P_i^{s+1} can then be reduced from the CSP. This particular reduction is not possible in planning graphs due to the presence of no-ops.
3. Variables that are forced to be true further reduce all variables that are mutex with them to be false. This does not help reduce a planning graph since in planning graphs mutexes are monotonically decreasing and, other than the initial state variables, variables are never forced to be true.[4] But in the CSP encoding the previous rules for forcing a variable to be true, and for creating a new mutex when it did not exist at the previous step, enable additional single valued variable reductions.

4.3 Sequence Constraints

There are some additional constraints that can be imposed on plans to rule out obvious inefficiencies. For example, it never makes sense in the plan to immediately follow an action by its inverse (e.g., a load followed by an unload). Adding this constraint allows eliminating different invalid combinations of actions by making mutex any inverse pairs of actions at consecutive time steps.

Blocking the immediate sequencing of inverse actions also allows us to infer additional sequence constraints. Say that actions A_i and A_j are inverses of each other, and that A_i is the only action that produces P_k while *Aj* is the only action that deletes it. Then another constraint that can be inferred from the mutex between A_i^s and A_j^{s+1} is that when P_k first becomes true, it must remain true for at least one more time step:

$$\neg P_k^s \wedge P_k^{s+1} \rightarrow P_k^{s+2}.$$

There is also a similar constraint for when $P_k$first becomes false.

Our system automatically detects action pairs that are inverses of each other as well as any predicates over which a sequence constraint can be imposed.

5 CSP-PLAN: Implementation

Based on the transformations explained before, we have developed a planner called CSP-PLAN that will find a solution to a planning problem. CSP-PLAN builds the BASE-CSP incrementally up to step *s* (beginning with *s* = 1), and then sends it to a CSP Solver for its solution. If the solver reports no solution to it, then the BASK-CSP is constructed for s+l. This cycle can continue until some termination condition has been reached or a plan has been found. It is important to notice CSP-PLAN solves these problems incrementally, thus ensuring that it finds a GP-step optimal plan.

The CSP solver used by CSP-PLAN is EFC [Katsirelos and Bacchus, 2001] which allows the use of different techniques

[4]This means that the variable that is mutex with the true variable never appears in the planning graph until the mutex no longer holds.

Problem	GAC	GACCBJ	GACvsCBJ	RecordNoGoods
Bw-large-12	0 46	0.10	0 10	0.11
Bw-large-a	1.44	0.49	0.49	0.50
Rocket-a	2166	13 74	1401	10 67
Rocket-b	5.26	4.44	4 48	3.39
DnverLog2	59 13	2109	21.28	1038
Driver Log3	0 42	0.23	0.23	0.15
Driver Log4	71.13	29.96	29 96	16.54
DrivcrLog5	139 98	62.98	63.65	29.59
DnverLog6	006	0.05	0.06	006

Table 1: Effect of changing the search algorithm on the performance of the CSP solver for planning problems. Experiments were run using the standard CSP heuristic dom+deg.

to solve a CSP problem, including: the ability to create and use tailored heuristics for dynamic variable ordering (DVO), the use of different state of the art CSP algorithms and a novel method to record nogoods. CSP-PLAN uses these methods as follows to efficiently solve general planning problems:

DVO: CSP-PLAN uses a dynamic variable ordering heuristic called BSEARCH, specially tailored for planning problems. This heuristic instantiates first the variables that propagate the most changes in the domain.

Search Algorithm: The CSP algorithm uses a version of GAC propagation [Bacchus, 2000] during backtracking along with nogood recording. In addition, for each constraint with an arity higher than 2, special purpose propagators were developed to increase the efficiency of GAC. It can be noted that although nogood recording is a generic CSP technique, in the form implemented it is strictly more general than the memoization of bad goal agendas described in [Blum and Furst, 1997].

6 Experimental Results

To test the performance of CSP-PLAN, we used STRIPS problems from IPC2 and IPC3 (International Planning Competition 2000/2002). All experiments run on a Xeon 2.40GHz machine with 400MB of RAM. Times are in CPU seconds.

6.1 Effect of CSP techniques on CSP-PLAN

Table 1 shows the effect of a few different CSP algorithms on the performance of CSP-PLAN on some representative problems (using a standard CSP variable ordering heuristic dom+deg).[5] We can see that GACCBJ and GACvsCBJ[6] perform over 2 times faster than plain GAC. GACCBJ and GACvsCBJ have comparable results on these problems, but we found that GACvsCBJ performed better on more complex problems, e.g., for Gripper-04, GACvsCBJ finds a plan in 877 secs while GACCBJ required 1,133 secs. The table also shows that nogood recording yields another factor of almost 2 improvement over GACvsCBJ. Hence, we choose GACvsCBJ+RecordNoGood as our default CSP search algorithm.

6.2 Effect of planning specific techniques

Some of the CSP simplification techniques explained in this article use planning related information, e.g, planning specific

[5]Other CSP algorithms were experimented with, but they did not provide adequate performance.

[6]GACvsCBJ is described in [Bacchus, 2000].

Problem	dom+deg	BSEARCH
Bw-large-12	0 10	0 08
Bw-lnrge-a	0.51	0.56
Rockct-a	10 65	0.54
Rockct-b	3 38	1.24
DnverLog2	10.38	0 46
DnverL.og3	0 15	0.03
DnvcrLog4	16.54	0.19
DriverLog5	24.59	0 17
DnvcrL.og6	006	0.01

Table 2: Effect of changing the DVO heuristic on the performance of the CSP solver for planning problems. Experiments were run using GACvsCBJ+RecordNoGoods

Problem	No redundant constraints	Inverse mutexes	Sequence constraints
Log-a	0.06	0 09	0 06
Log-b	16.14	15.17	213.50
Rockct-a	1 41	148	1.31
Rockct-b	1.85	1 77	2 15
Gripper-01	0 12	0 06	0 10
Gripper-02	17 11	13 20	1701

Table 3: Effect of adding mutexes between inverse actions and sequence constraints to the BASE-CSP.

DVO heuristics, the addition of redundant constraints, the removal of single valued variables, etc. The addition of redundant binary constraints, particularly mutexes, and the detection and removal of single valued variables has a clear and dramatic effect on the efficiency of CSP-PLAN, even for simple problems. For example, on bw-large-12, the BASE-CSP has 1,289 variables and 10,426 constraints, but only 657 variables and 2,847 constraints after single valued variables are removed. Similarly, without the inferred mutex constraints and the removal of single valued variables, CSP-PLAN required 154.3 sec. to solve the problem. Adding the mutex constraints reduced this time down to 0.33 sec, and removing single valued variables brought the time down even further to 0.04 sees. The effect of some of the other planning specific techniques is described below.

DVO heuristic: The order in which variables are instantiated during search greatly affects the solution times for a CSP. Table 2 shows the difference in performance between the standard CSP heuristic **dom+deg,** and our BSEARCH heuristic which was designed specifically to take advantage of the planning based structure of the CSPs that CSP-PLAN is solving. These representative problems show that BSEARCH performs much better than dom+deg.

Redundant Constraints: Table 3 shows the performance of CSP-PLAN when mutexes between inverse actions and sequence constraints are added to the BASE-CSP. On these problems we see that there is a slight improvement in CSP-PLAN when mutexes between inverse actions are present. Interestingly, adding sequence constraints did not help. Minor improvements on some problems were negated by significant degradation on others.

6.3 Comparison with similar approaches

The results in Table 4 show how CSP-PLAN (using the same setting for all problems)[7] compares with GRAPHPLAN [Blum

[7] A number of planning domains could be solved faster with different settings; e.g., the heuristics sometimes performed differently

Problem	CSP-PLAN	GKAPHPLAN	Gr-Csp	BLACK BOX	IPP(4 1)
Dw-largc-a	0.18	021	0.63	0.23	0.06
Bw-large-b	15.18	19 53	106.00	17.50	2.84
Gripper-02	15.36	2.17	62 00	1 87	0.10
Gripper-03	1.328 63	135 00	> 3hn».	2,485 00	1.08
Log-a	0.05	1,733.00	47 75	5 50	165 20
Log-b	14.87	675.00	143.00	5.72	120 64
Rocket-a	1.51	46 46	40 50	5 44	5 96
Roekct-b	1.79	128.00	22.68	5.57	21 10
Depots3	1103	241	15 75	2 36	0.22
Depots4	4.59	3.80	82.00	3 72	0.44
DrivceLog8	0.78	16 05	0.38	5 70	2.67
DriverLog9	28 99	270 00	90.00	6.27	6.50
DriverLog10	77 75	123 00	22.23	7.89	11 08
ZcnoTrave16	0.24	0.64	0.55	0 43	CNF
ZenoTravel7	0.18	0 67	0 61	0 44	CNP
ZenoTravel8	0.52	140	124	0 90	CNP
ZenoTravel9	37 16	4.84	4.10	3.40	CNP
ZenoTravel 10	3.55	9,228 00	28 53	7.52	CNP
Satellite 3	0 13	0 23	0 20	0 23	0.04
Satclhle4	5 08	271.00	77.02	6.26	4.83
Satellite5	262 72	9,256 00	5.759 00	6.49	175 27
Freccelll	0.10	14 24	183 00	12 90	0 88

Table 4: Comparison of CSP-PLAN with GRAPHPLAN, GP-CSP, BLACKBOX and IPP on a set of problems from the STRIPS domain. "CNF' means could not parse. Lowest times are in bold.

and Furst, 1997], GP-CSP [DO and Kambhampati, 2001], BLACKBOX [Kautz and Selman, 1998] and IPP 4.1 [Koehler *et al,* 1997]. These planners have in common the construction of a planning graph. The planning graph is either then compiled into a CSP (GP-CSP) or a SAT formula (BLACK-BOX), or searched directly for a solution (GRAPHPLAN and IPP 4.1). CSP-PLAN, on the other hand, bypasses the planning graph construction and directly exploits the CSP encoding of the problem as explained above.

It can be seen that CSP-PLAN yields significantly better performance in almost all cases over the standard planning graph planners GRAPHPLAN and GP-CSP. This provides evidence of the effectiveness of the additional constraints and reductions computed by CSP-PLAN. The results against BLACKBOX are mixed. This version of BLACKBOX utilizes the highly engineered Zchaff [Moskewicz *et aL,* 2001] SAT solver. ZchafTutilizes very different heuristics and techniques for learning nogoods (clauses) than CSP-PLAN and is more efficient that our current CSP code on larger problems. Nevertheless, there is considerable scope for making our CSP-PLAN implementation more efficient, and our heuristics are at this stage very simple. Even so, our approach demonstrates its usefulness on problems like gripper.

Similarly against IPP our results are mixed. A key component of IPP is a method for solving planning problems by dividing them into a set of smaller problems using a goal agenda [Koehler, 1998b]. This technique is particularly effective on domains like BlocksWorld, Depots and Gripper, and it can easily be recast in terms of CSPs. Hence, CSP-PLAN could be extended to utilize this technique. On other domains like logistics CSP-PLAN is more effective. This is due particularly to the presence of mutexes between inverse actions in these domains that can be effectively used by CSP-PLAN.

on different domains.

MirsOpt Level	Time	Steps	Acts	Res
1	3 52	110	175	11.761
3	7 09	102	178	11.505
5	28 02	108	197	13,130
7	80 44	105	221	10.853
9	96 44	104	225	11,630
C'SP-PI AN	774 86	75	194	11,345

Table 5: Effect of changing the optimization value for MIPS. Each value in this table is the sum of the results for Driver-Log 1 to DriverLoglO. "Res": resource usage, "Steps": number of GP-steps, and "Acts": number of actions in the plan.

7 Extensions: Planning with Resources

A major advantage of encoding planning problems as CSPs is that we can easily extend the encoding to represent more complex planning problems. For example, planning problems with resources can simply be represented as CSPs with numeric variables. To demonstrate the flexibility of our approach we implemented an extension of CSP-PLAN to deal with such planning problems. Previous approaches based on planning graphs (e.g., [Koehler, 1998a]) have had to develop from scratch methods for dealing with the resources variables in the planning graph, like interval arithmetic and bounds propagation. These techniques are already well known in the CSP field.

To handle planning with resources we introduced the following into CSP-PLAN: (1) Mathematical operation constraints, e.g., $x = z + y$; (2) comparison constraints, e.g., $x > y$; (3) precondition constraints containing comparisons, e.g., fuel (tru > 0; (4) successor state constraints to define how the numeric variables could change between states, e.g., if no action affecting it is true, its value cannot change; (5) and finally, additional concurrency constraints to block actions from simultaneously altering a numeric variable. These new constraints were very easy to add to the BASE-CSP, and the reduction transformations described above were used almost unchanged to simplify the BASK-CSP.

We compared CSP-PLAN with MIPS [Edelkamp, 2002][8] to evaluate its performance on resource planning problems. MIPS combines two paradigms, model checking and heuristic search, to find totally ordered plans that are then made concurrent using a post-processing procedure. This combination of paradigms introduces additional complexity to the algorithm, in contrast with the uniform paradigm used in our approach. MIPS tries to minimize the number of steps in the final plan, using an optimization parameter that ranges from 1 to 10. Table 5 shows this how parameter affects the quality of the final plans in MIPS. Increasing the parameter causes MIPS to spend more time trying to find plans with fewer steps. However, this search is not always successful. In fact MIPS can spend more time and generate worse results. CSP-PLAN, on the other hand, is an optimal planner always finding the shortest plan in terms of number of GP-steps. This is a sightly different optimization criteria from that used by MIPS. However, even under MIPS's optimization criteria CSP-PLAN\S plans are much better than those found by MIPS.[9]

Table 6 compares both planners (MIPS running with its optimization flag set to 5). The results in this table show that

[8]MIPS competed in IPC3. No planning graph based planner was

	CSP-PLAN				MIPS			
Problem	Time	Steps	Acts	Res	Time	Steps	Acts	Res
Depotsl	0 05	7	11	32	0 04	9	11	32
Depots2	1.55	10	16	43	0 06	10	18	44
DriverLogl	0.01	6	8	1099	0.04	6	8	1099
DriverLog2	0 47	9	23	979	0.08	12	22	1605
DriverLog3	0 08	7	12	907	0 04	7	14	1051
DriverLog4	0.13	7	18	703	5 77	9	19	757
DriverLogS	0 21	8	18	654	121	13	22	838
DnvcrLog6	0.04	5	13	965	0.15	7	15	1667
Driverl.og7	0 14	6	18	1060	0 25	7	13	866
DriverLog8	5 24	8	27	1920	15.78	17	29	2217
DriverLog9	73.27	1!	32	2485	1.67	14	30	2787
Driver Log10	695.27	8	25	573	3.03	16	25	243

Table 6: Performance comparison of CSP-PLAN and MIPS on resource planning problems. "Res", "Steps" and "Acts" as explained in table 5.

CSP-PLAN has a performance comparable to MIPS, and in most cases, the quality of the plans (number of steps) is considerably better. However, CSP-PLAN does not scale as well as MIPS, e.g., it takes much longer than MIPS on DriverLog l0. But is not surprising since CSP-PLAN is an optimal planner while MIPS is not. Achieving optimality quickly becomes very difficult. It is also important to note that our prototype implementation uses very simple techniques to propagate numeric variables. These techniques are far from the current state of the art in CSPs. Nonetheless, we see that we still obtain results competitive with the competition version of MIPS.

8 Conclusions

Our implementation of CSP-PLAN has demonstrated that the structure of planning problems can be exploited directly in a CSP encoding. By lifting the planning problem to a CSP we obtain a richer and more robust representation for which many sophisticated and effective solution techniques have been developed. In particular, we are able to capture and generalize the important inferences generated by planning graphs directly in the CSP representation. This provides excellent performance for CSP-PLAN in comparison with similar techniques. Additionally, by showing how easily we can extend our representation to planning with resources, we have provided evidence for the benefits of lifting the problem to a CSP encoding.

This work naturally leads to a number of future research topics. For example, the fastest current planners are based on local search. Nevertheless, the heuristics used in these planners utilize notions from the planning graph construction. It seems feasible that instead of planning graphs, CSP techniques could be utilized directly to compute better heuristics. This could be particularly beneficial for computing more informative admissible heuristics. Better heuristics and additional transformations could also be explored. In particular, there is still much scope for improving the performance of this approach. Finally, the approach is well suited to resource usage optimization using branch and bound techniques during the search, and this could be further investigated.

extended to deal with resources in this competition.

[9]Note that neither planner is actually trying to optimize resource usage, just the size of the plan.

References

[Bacchus and Ady, 2001] Fahiem Bacchus and Michael Ady. Planning with resources and concurrency, a forward chaining approach. In *IJCAI-2001,* pages 417-424.

[Bacchus, 2000] Fahiem Bacchus. Extending forward checking. In *CP2000,* number 1894 in LNCS, pages 35-51. Springer-Verlag.

[Blum and Furst, 1997] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. *Artificial Intelligence,* 90:281 300, 1997.

[Do and Kambhampati, 2001] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence,* 132(2):151-182, 2001.

[Edelkamp, 2002] Stefan Edelkamp. Mixed propositional and numerical planning in the model checking integrated planning system. In *AIPS-2002, Workshop on Temporal Planning,* 2002.

[Freuder, 1985] E. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM,* 32(4):755-761, 1985.

[Getoor *etai,* 1997] L. Gctoor, G. Ottosson, M. Fromherz, and B. Carlson. Effective redundant constraints for online scheduling. In *AAAI-1997,* pages 302 307.

[Katsirelos and Bacchus, 2001] George Katsirelos and Fahiem Bacchus. A library for solving CSPs, 2001. www.cs.toronto.eduTgkatsi/efc.tar.gz.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *AAAI-1996,* pages 1194 1201.

[Kautz and Selman, 1998] H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. Workshop on Planning as Combinatorial Search, AIPS-1998.

[Kochler *et al,* 1997] J. Kochler, B. Ncbcl, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *European Conference on Planning,* pages 273 -285, 1997.

[Koehler, 1998a] J. Kochler. Planning under resource constraints. In *ECAI-1998,* pages 489-493.

[Kochler, 1998b] J. Koehler. Solving complex planning tasks through extraction of subproblems. In *AIPS-1998,* pages 62-69.

[Moskewicz <7 *al.,* 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proc. of the Design Automation Conference (DAC),* 2001.

[Pednault, 1989] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *KR-1989,* pages 324-332.

[Pirri and Rciter, 2000] F. Pirri and R. Reiter. Planning with natural actions in the situation calculus. In Jack Minker, editor, *Logic-Based Artificial Intelligence.* Kluwer Press, 2000. in press.

[Reiter, 2001] R. Rciter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[van Beek and Chen, 1999] P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *A A AI-1999,* pages 585-590.

[Vossene/a/.,2001] T. Vossen, M. Ball, A. Lotem, and D. Nau. Applying integer programming to AI planning. *Knowledge Engineering Review,* 16:85 100, 2001.