ECAI 2010
H. Coelho et al. (Eds.)
IOS Press, 2010

doi:10.3233/978-1-60750-606-5-453

Constraint Based Planning with Composable Substate Graphs

Peter Gregory and **Derek Long** and **Maria Fox**
University of Strathclyde
Glasgow, UK
firstname.lastname@cis.strath.ac.uk

Abstract.

Constraint satisfaction techniques provide powerful inference algorithms that can prune choices during search. Constraint-based approaches provide a useful complement to heuristic search optimal planners. We develop a constraint-based model for cost-optimal planning that uses global constraints to improve the inference in planning.

The key novelty in our approach is in a transformation of the SAS$^+$ input that adds a form of macro-action to fully connect chains of composable operators. This translation leads to the development of a natural dominance constraint on the new problem which we add to our constraint model.

We provide empirical results to show that our planner, Constance, solves more instances than the current best constraint-based planners. We also demonstrate the power of our new dominance constraints in this representation.

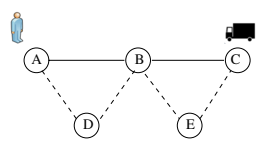
1 INTRODUCTION

Planning is a combinatorial optimisation problem. Hooker [16] has observed that solutions to such problems combine search, inference and relaxation. In cost optimal planning, search and relaxation techniques have dominated, with traditional A* search, and variants of it, using informed but admissible heuristics proving to be most effective [14]. In general, inference has played a less significant role: reachability analysis is used to infer action choice constraints, mutex reasoning has been exploited to support some propagation between action choices and there has been some work on symmetry reduction [9]. In contrast, in the constraint reasoning community inference plays a central role, with constraint propagation techniques typically far outweighing the use of relaxation in solving finite-domain constraint satisfaction problems (CSPs).

There have been several attempts to exploit CSP technology in planning. CPT [21], GP-CSP [6] and SeP [1] are examples of cost optimal constraint-based planners. The key to successful use of CSP in planning is in developing planning models that exploit the inference mechanisms CSP offers. CPT, in particular, demonstrates that with a carefully crafted model it is possible to solve a significant proportion of planning benchmarks without any search at all, purely by exploiting propagation of constraints.

In this paper, we further develop this line of domain-independent planning research, exploring the use of a CSP encoding of planning problems that is based on the SAS+ [4] representation, but incorporating a new extension using automatically derived *macros* and exploiting a collection of constraints that these must satisfy. These constraints support further inference and, as we demonstrate using Constance, our implemented solver, offer performance enhancement for CSP-based cost-optimal planning.

Throughout this paper, we make use of a simple running example. This is shown in Figure 1 and is an example instance of the Driverlog domain. The driver in this example can walk along dotted lines (footpaths), but not along solid lines (roads). The truck in the example can only drive along solid lines. Drive actions require the driver to be aboard the truck.



Variables

Driver {A,B,C,D,E,Truck}
Truck {A.B.C}
TruckOcc {True,False}

Initial State and Goal

{<Driver,A>,<Truck,C>,<TruckOcc,False>}
{<Driver,B>,<Truck,B>}

Action: (WALK A D)
Prevail: {(path A D)}
Pre/Post: {<Driver,A> ➛ <Driver,D>}

Action: (DRIVE A B)
Prevail: {(road A B), <TruckOcc,True>}
Pre/Post: {<Truck,A> ➛ <Truck,B>}

Action: (BOARD A)
Prevail: {<Driver,A>,<Truck,A>}
Pre/Post: {<Driver,A> ➛ <Driver,Truck>, <TruckOcc,False> ➛ <TruckOcc,True>}

Figure 1. A simple planning problem, and the corresponding SAS$^+$ representation, used as a running example. The initial state is shown to the top-left and the goal state to the top-right.

2 BACKGROUND

The widely adopted planning domain language, PDDL [8], is a propositional language. This makes its translation into Planning-as-Satisfiability encodings straightforward, but it is less effective as a basis for CSP encodings. SAS+ [4] encodings have become of increasing interest and Helmert has shown that it is possible to automatically translate from PDDL into SAS+ for a large fragment of PDDL [13]. This encoding is well-suited to CSP encodings, since it relies on variables with multi-valued domains, rather than purely boolean domains, offering opportunities for the propagation techniques used in CSP solvers to demonstrate their power and also leading to a compact representation of planning problems compared with

the grounded propositional form. Part of the SAS+ encoding of our running example is shown in Figure 1.

Representing planning problems in SAS+ also leads to the identification of domain transition graphs [12] (DTGs), which capture the legal transitions that are possible between assignments to the same variable at successive time points as labelled directed graphs. The arcs of each of these graphs are labelled with the actions and the vertices with the values of one of the variables. Figure 2 shows the DTG for the driver variable in our example.

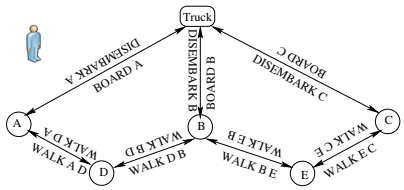


Figure 2. A domain transition graph for the driver variable from the example in Figure 1.

2.1 Cost-Optimal Planning

The optimal planning track in the 6th International Planning Competition focussed on sequential-optimality, with action costs. This track used a small extension to PDDL, in which one specific numeric fluent records accumulated action costs, with each action adjusting this fluent as it executes. Using this fluent and actions with fixed and positive costs, it is possible to unambiguously define the cost-optimal plans for a planning problem as the plans whose total action cost is least across all plans for the problem. In a classical STRIPS problem, each action is taken to have a unit-cost, and so the optimal plan is the plan with the fewest actions.

In the most recent International Planning Competition the baseline planner (A* search with a zero-valued heuristic) performed better than many of the optimal planners.

2.2 Constraint Satisfaction Problems

In a CSP, finite-domain variables are constrained in the values that they can legally take simultaneously. A solution to a CSP is a full assignment to the variables such that no constraints are violated.

Constraint Programming solvers rely on powerful propagation algorithms during search in order to filter inconsistent values from the domains of the variables during search. Constraint Programming can be particularly effective when global constraints can be used. Global constraints act over many variables at the same time. For example, a set of variables that need to have distinct values can be constrained by the *all-different* constraint. Another example is the *table* constraint [10], in which the valid combinations of values for a set of variables are specified explicitly. This type of constraint is useful when the number of valid combinations of assignments are small with respect to the total number of valid assignments.

It is important to recognise that global constraints are not simply a syntactic convenience. Specialised propagators can dramatically prune the search space at low computational cost.

3 CONSTRAINT BASED PLANNING

Two CSP-based planners have been developed in recent work: CPT [21] and SeP [1]. CPT is a cost-optimal, temporal, partial-order planner. CPT uses the PDDL planning formalism. In the CPT constraint model, variables represent facts to be achieved. The domains of each of these variables correspond to the actions that can achieve the fact. CPT uses admissible heuristics to impose a lower-bound on the plan length during its search. It also uses landmark information [15] to provide clues on ordering actions. Standard finite-domain CSPs do not have sufficient expressiveness to capture planning problems directly. Instead, they are used to represent a succession of bounded problems, iteratively increasing the bound until a solution is found. In CPT, the bound is placed on the number of distinct copies of any single action within the plan. Initial analysis can place a lower bound on this number, but subsequent search must iteratively explore increasing values for these bounds until a plan is found (it is possible to place an upper bound, too, but this is exponential in the size of the problem in general, so of little practical interest).

SeP follows a more common representation in which the bound is placed on the plan length and variables are used to represent the choice of action at each step in the plan up to this bound. This approach is also used in GP-CSP [6] and in Planning-as-Satisfiability [18]. SeP is a sequential optimal (non-temporal) planner for uniform action costs, effectively minimising the number of actions.

SeP uses a model (Figure 3) in which CSP variables at a given time-point correspond to the SAS$^+$ variables at that time-point. Given a SAS$^+$ task, $P = \langle \mathcal{V}, \mathcal{O}, v_0, v_\star \rangle$, and a specified plan length, T, a SeP constraint model contains $T+1$ state variables corresponding to each variable in $V \in \mathcal{V}$, recording the value of V at each time-point in the plan, and a set of T variables that represent the actions that occurs at each point between successive states.

In general, the variables that represent sequences, either of SAS+ variable assignments or of actions, can be thought of as timelines describing the behaviours of each of the variables throughout the plan. It is in terms of timelines that we define the general SeP instance, $T = \langle \mathbf{Act}, \mathbf{Var} \rangle$, where $\mathbf{Act}$ is the action timeline and $\mathbf{Var}$ is the set of variable timelines. We refer to $\mathbf{Act_t}$ to mean the action at time t. We refer to $\mathbf{Var_{v,i}}$ as the value of variable V at time t.

SeP uses table constraints to ensure that the effects of action transitions, prevail conditions and frame axioms are correctly enforced between layers. These table constraints are a generalisation of the simpler trajectory constraints used in CSP-Plan [19]. A table $\mathbf{Tab}_V$ is constructed for each $V \in \mathcal{V}$ such that for all actions, a:

- If $\langle V, v \rangle$ is part of the prevail conditions of a:
  The row $\langle a, v, v \rangle$ is one of the allowed tuples in $\mathbf{Tab}_V$
- If $\langle V, p, e \rangle$ is part of the pre/postconditions of a:
  The row $\langle a, p, e \rangle$ is one of the allowed tuples in $\mathbf{Tab}_V$
- If V is neither part of the prevail conditions nor the pre/postconditions of a:
  For all values $v \in Dom(V)$, the row $\langle a, v, v \rangle$ is contained in $\mathbf{Tab}_V$

Then, for all pairs of successive time-points, $\langle t_i, t_{i+1} \rangle$, the table $\mathbf{Tab}_V$ is used to constrain the possible assignments to variables $\mathbf{Act}_t$, $\mathbf{Var}_{V,t}$ and $\mathbf{Var}_{V,t+1}$. Describing action transitions in this way allows more inference to be performed at each search node.

These table constraints are sufficient to allow search and inference to arrive at a plan. However, to improve performance of CSP solvers the trick is to find additional constraints that can be added to the model that will allow more direct pruning of the search space and propagation of inferences through the model. SeP therefore supplements these basic model constraints with dominance constraints to reduce the search-space further. These constraints depend on the

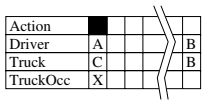


Figure 3. A timeline representation of the example planning problem. The first action cell is not used because there is one more state than actions.

well-known concept of interfering actions: two actions are said to interfere if they have competing needs (delete the same precondition) or competing effects (achieve different effects for the same variable). This definition is the same as in Graphplan [2] generalised to SAS+ variables.

If two non-interfering actions occur in successive time-points in the plan, then the order in which they appear does not alter the outcome of the plan. Therefore, they can be ordered arbitrarily. For all non-interfering actions, a_1 and a_2, an ordering is selected arbitrarily (say $a_1 \leq a_2$) and a constraint is added to the model that, for all pairs of successive time-points t and $t+1$, $\mathbf{Act}_t = a_1 \rightarrow \mathbf{Act}_{t+1} \neq a_2$. This constraint can be represented as a negative table constraint (in which the *disallowed* tuples are specified explicitly). It is worth noting that this constraint is a form of symmetry reduction, eliminating plan permutation symmetries that arise when different orderings of actions achieve the same outcome.

SeP searches in a forwards direction, using no lower bounding information: it begins by searching for a plan of length 1 and increments the plan length as it proves a level infeasible.

4 IMPROVING THE CONSTRAINT MODEL

The model used by SeP forms the starting point for our own work: we begin by using an automatic translation of a PDDL model into a SAS+ encoding, using Helmert's translator [13], then construct a CSP model based on the timelines for the SAS+ variables and for actions. However, we perform a further step of analysis on the model to extract a collection of *macros* that can be safely added to the CSP model, together with the constraints required to manage their use while retaining correct cost-optimal plans. The macros allow us to exploit a further set of dominance constraints and lead to stronger pruning in the CSP models.

4.1 Connected Substate Groups

Macro-actions have been recognised as a useful tool in planning for some time [3, 5, 17]. The essential idea is that some actions naturally work together in recurring patterns that can be most efficiently constructed just once and then reused, rather than reconstructed repeatedly from the primitive action components. A variety of techniques have been used to identify candidate patterns and then to exploit them. In general, exploitation is hard because it is usually necessary to leave the primitive action choices available in the search space in order not to prevent the planner from finding solutions, but at the same time it is important to encourage the planner to use macros where possible. It turns out that we can achieve both objectives straightforwardly within the CSP models we construct.

In our work macros are constructed by identifying sequences of actions that compose with one another under common prevail conditions. A simple case for our running example is shown in Figure 4. The condition for two actions to compose is as follows:

Definition 1 (Action Composition) *Actions a and b compose if the union of the postcondition and prevail conditions of a, a^+, and the precondition of b, b^-, satisfy $a^+ = b^-$. The composition of a and b is a new action c with precondition a^- and postcondition b^+, and cost equal to the sum of the costs of a and b.*

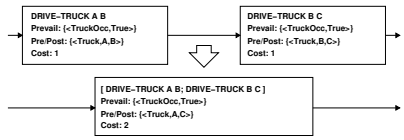


Figure 4. An example of a composing action macro. Two cost 1 drive actions compose into a cost 2 drive macro-action.

In order to discover and make use of composable action macros, we create a structure that we call a substate graph. This is a graph between preconditions and effects of operators.

Definition 2 (Substate Graph) *Given a SAS$^+$ task $P = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, a substate graph $G_{sub}(P) = (V, E)$ is a weighted, labelled, directed graph such that V comprises O^- and O^+ for all $O \in \mathcal{O}$. For all operators $O \in \mathcal{O}$, an edge exist between the two nodes O^- and O^+. The weight placed on the edge is the cost of O, and the edge is labelled O.*

By finding the strongly connected components of this graph, we reveal all of the composable action macros in the problem. If two unconnected nodes belong to the same connected component, then the shortest path between them defines the least-cost composable action macro between substates. These macro actions are added to the model.

Definition 3 (Connected Substate Group) *Given a substate graph, $G = (V, E)$, let the strongly connected components of G be labelled $c_1, ..., c_n$.*

The connected substate function maps each operator O to the label of the connected component containing the edge labelled O.

An important observation is that once any substate in this graph is achieved then an arbitrary length sequence of composable actions can be performed within the connected component that contains that substate. This is because we know that there is a path between all substates in the connected component. As long as only actions from the same component are performed, there remains a path to all other substates in that component. Once a prevail condition is achieved, any sequence of actions that rely on that prevail can be performed. For example, once a driver has boarded a truck then any number of drive actions can be sequenced. In our running example, the substate groups are formed for the driver (who can walk between any pairs of locations as a single macro action), for the truck (which can drive between any pair of locations as a single macro action, once the driver is on board) and for the driver-truck pair at each location (where the driver can board or disembark the truck).

4.2 Substate Groups and Optimal Planning

Connected substate groups can also be used to improve constraint propagation in optimal planning. Each action in an optimal plan *must* contribute to the achievement of a goal in some way. In other words, at least one of the effects of an action must be either a precondition or prevail condition of a later action or a final goal.

Definition 4 (Dependent Action) *Action a_2 is dependent on action a_1 if at least one of the effects of a_1 is a precondition of a_2.*

Theorem 1 *Given a cost-optimal plan of shortest length, $p = a_1, ...a_n$, containing two actions a_i and a_j $(i < j)$ from the same substate group, c, there must exist an action a_d in the sequence $a_{i+1}, ..., a_{j-1}$ such that a_d is dependent on a_i.*

Proof: Assume that there are no actions dependent on a_i between a_i and a_j. Actions a_i and a_j compose, since they are from the same substate group. If no actions between a_i and a_j depend on a_i, then a_i can be replaced by the composition of a_i and a_j, and a_j can be removed from the plan. Since this plan has fewer actions than p, p cannot be an optimal plan with the fewest actions.

4.3 The Constance Constraint Model

The analysis we have now described can help form constraints that lead to greatly improved pruning of the search tree in a sequential-optimal constraint based planner. Theorem 1 leads us to the rule that after a transition has been made, then no transition in the same substate group can be made until one of the values in the effects of the transition has been used.

We construct a timeline constraint model similar to that used within SeP, extended to handle action costs and substate groups. In addition to the action timeline and variable timelines, cost and substate group timelines are added. In order to bind the actions, groups and costs, we create a table constraint with each row representing a triple, $\langle a, g_a, c_a \rangle$, where a is the action, g_a represents the substate group for action a and c_a representing the cost of action a.

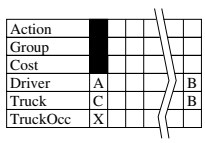


Figure 5. Constance timelines, including costs and substate groups.

We provide the same constraints as SeP, to handle the action effects and frame axioms, including the dominance constraints defined in SeP. We also provide another powerful dominance constraint set, which exploits Theorem 1. It is apparent from that result that a constraint on optimal plans is that no action from the same group as an action already appearing earlier in the timeline can be applied without an intervening action that depends on the first action. There are various ways this constraint can be imposed. Following careful experimentation, we approximate the constraint by preventing actions from the same substate group from being performed in two successive action steps. This is a weaker constraint than the theorem implies, but we found it to be a good tradeoff between propagation costs and pruning impact.

5 THE CONSTANCE PLANNER

The Constance planner uses two alternative search strategies: forwards and backwards. The forward search strategy uses an initial lower bound of length one, and then increments the plan length by one each time a level is proven inconsistent. Once a solution is found (with cost c), it is not guaranteed optimal (since action costs need not be unit), so the horizon is increased to c divided by the minimum action cost and optimised, using c as an upper bound.

Our backward strategy, similar to that used in the Meta-CSP SAT planner [11], uses the sub-optimal planner, LAMA [20], to find an upper-bound, u, on the optimal plan length. We then search backwards from a horizon set using u divided by the minimum action cost. We add a *noop* action, which has zero cost and does not alter the state, with constraints preventing this action from being followed by any non-noop action. This allows the planner to find the optimal plan without constructing and searching multiple models, by padding the end of the plan with noop actions. A problem arises when the minimum (non-noop) action cost in the domain is zero. In this case, which occurs for example in Sokoban from the Sixth IPC, we cannot guarantee optimality. We can, of course, still provide either step optimality, or cost optimality for a given horizon.

The Constance planning system implements the following procedure:

1. Translate the PDDL planning model to produce the SAS$^+$ variables and operators [13].
2. Construct substate graphs from the SAS$^+$ structures.
3. Find strongly-connected components of substates graphs.
4. Extract the composing action macros.
5. If searching backwards, then run LAMA on the original SAS$^+$ problem to find an upper-bound.
6. Construct and solve the constraint model.
7. If no solution is found and searching forwards, then increment the horizon and return to the previous step.

6 EMPIRICAL EVALUATION

All experiments are performed on a desktop PC with a Intel 3.16GHz CPU, a 2GB memory limit and a 30 minute time cutoff. We use SICSTus Prolog 4.0.8 to generate the SeP results. Constance is implemented in Java, using the Choco Constraint Programming library (version 2.0.0.3). We use the Sun 1.6 (version 16) Java Virtual Machine to generate the Constance results. We compare our forwards and backwards search strategies against CPT and SeP. All timings, for SeP and Constance include translation time from PDDL.

SeP uses an intermediary Prolog input format. We have implemented a translator from SAS$^+$ to this intermediary representation in order to form useful comparisons.

6.1 Results

Table 1 shows the number of instances solved for the first ten instances of IPC benchmark domain sets. We restrict attention to this set because none of the planners solve problems above this cut off, with the exception of Blocksworld, which we examine further below. We total the number of instances solved by each planner at the bottom of the table. We also show the number of domains for which each planner solves most instances. The two Constance variants solve at least equal greatest number of instances in 15 of the 18 domains, and are outright winners in ZenoTravel, TPP, Pipesworld NoTankage, Pipesworld NoTankage, FreeCell and DriverLog. SeP dominates in Airport, Openstacks and Pathways.

Figure 6 shows the time performance in three of the domains. For these domains, we extend the analysis to all of the instances from the domains. We selected the domains Blocksworld, Airport and Pipes-NoTankage as all three planners solve a reasonable number of instances from these domains. Another reason we select these domains in that each planner dominates in one of the domains.

Blocksworld seems particularly suited to the CPT model rather than the timeline based planners. Another interesting domain is Openstacks, in which SeP is the only planner to solve any instances. Solving five instances brings SeP's performance close to the best heuristic A* planners [14] on this domain.

In all domains, the results show little difference between the forward and backward search approaches.

Domain	CPT	SeP	Con$_f$	Con$_b$
Airport	6	**8**	6	7
Blocks	**10**	**10**	**10**	**10**
Depot	**2**	**2**	**2**	**2**
Driverlog	3	3	**6**	**6**
FreeCell	0	3	**4**	**4**
Grid	1	**2**	**2**	1
Gripper	**2**	**2**	1	**2**
Logistics 2000	5	**6**	**6**	**6**
Logistics 1998	**0**	**0**	**0**	**0**
Miconic STRIPS	**10**	**10**	**10**	**10**
Openstacks	0	**5**	0	0
Pathways	3	**4**	3	3
Pipesworld NoTankage	4	6	**9**	**9**
Pipesworld Tankage	2	2	**7**	**7**
PSR-Small	**10**	**10**	**10**	**10**
Rovers	**4**	**4**	**4**	**4**
TPP	4	4	**5**	**5**
ZenoTravel	4	5	**8**	**8**
Total (instances)	70	86	92	**93**
Total (domains)	7	12	**14**	**14**

Table 1. Performance of constraint based planners, showing numbers of instances solved amongst the first ten instances in each domain. Con$_f$ and Con$_b$ refer to Constance in forward and backward modes.

7 ANALYSIS

The results show that in domains with large substate groups (Driverlog, ZenoTravel, Pipesworld, TPP) Constance markedly improves the performance of constraint based planners. These domains tend to be transportation-type domains, where the underlying maps provide the large composable substate graphs. The Freecell results show that the approach is not simply beneficial to transportation domains.

We expect SeP to perform better than Constance in domains with few composable action groups given that computation of the groups imposes an overhead before search begins and managing the group variables imposes additional cost during constraint solving.

Table 2 shows that the relative benefits we obtain from using macros and from the dominance constraints they support varies across domains. In some domains, such as TPP, the macros themselves do not help, but the dominance constraints they support lead to significant benefits. In other domains, such as Driverlog, the macros already account for much of the performance improvement. In all cases, the dominance constraints enhance the use of macros.

The one domain in which CPT performs much better than either SeP or the Constance variants, BlocksWorld, is interesting. CPT solves more of these instances than the current state of the art optimal planners [14], while SeP and Constance cannot solve many of them in 30 minutes. In these instances, CPT makes only a small number of choices over which actions are needed to support open preconditions. This is probably because CPT constructs its plans as partially ordered structures, which allows it to interleave the block reordering

	No Macros		Macros		Banned Groups	
Instance	time	nodes	time	nodes	time	nodes
tpp01	0.68	1	0.60	1	0.69	1
tpp02	1.10	19	1.11	19	1.10	15
tpp03	2.02	256	2.06	256	1.98	133
tpp04	4.07	5507	4.09	5507	3.00	1406
tpp05	749.53	1100520	–	–	505.29	676972
zeno01	1.25	1	1.64	1	1.44	1
zeno02	1.98	12	2.71	11	2.39	11
zeno03	3.37	33	4.40	61	4.65	44
zeno04	3.98	234	5.08	39	5.32	37
zeno05	13.80	4176	25.72	3874	25.22	3014
zeno06	47.23	19105	79.50	16165	56.78	9529
zeno07	–	–	181.52	37828	121.20	19306
zeno08	104.14	9694	168.34	6164	157.66	4491
driverlog01	2.12	6	1.95	5	2.22	5
driverlog02	–	–	252.73	252250	147.28	138122
driverlog03	8.39	7531	4.64	1487	4.28	1267
driverlog04	509.17	476407	60.81	40773	50.38	38260
driverlog05	–	–	–	–	–	–
driverlog06	26.92	12290	5.78	629	5.56	442
driverlog07	387.36	153362	100.50	33861	68.32	21123
driverlog08	–	–	–	–	–	–

Table 2. Performance comparisons for Constance without macros, with macros and with both macros and substate group dominance constraints.

actions as the plan develops. The timeline approach forces commitment to the timing of actions as they are added to the plan and this choice is hard to get right without landmark information in Blocks problems.

We hypothesise that a model combining a timeline and a partially ordered model, connected by channelling constraints, could lead to a planner that benefits from the advantages of both models.

Domain		CPT	SeP	Con$_f$	Con$_b$	h^{C}	G
Airport	(50)	7	11	10	10	**38**	11
Blocks	(35)	**34**	10	13	13	28	30
PipesNT	(50)	4	6	9	9	**17**	11

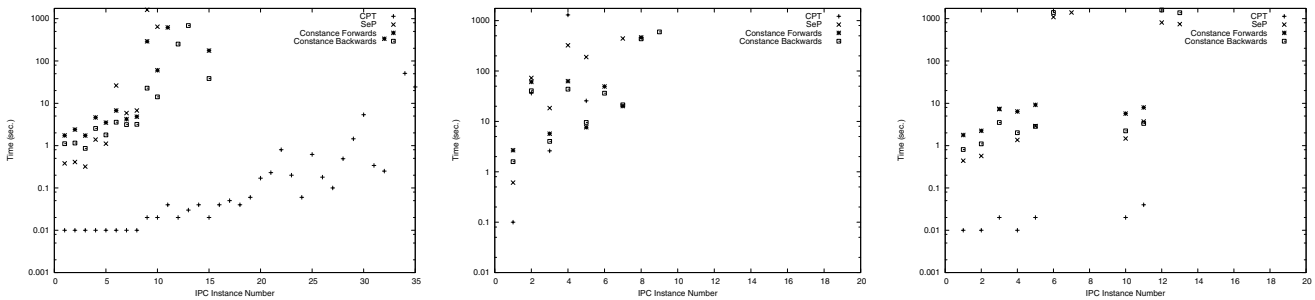
Table 3. Comparison with heuristic state-space planners: planners CPT, SeP, Constance Forwards, Constance Backwards, Landmark Cut Heuristic and Gamer; cells showing the number of instances solved with a 30 minute cut-off.

We provide a comparison with the current best optimal planners [14] in Table 3. The Landmark Cut and Gamer [7] results are taken from [14]. Performance of constraint-based planners is still below that of heuristic planners.

8 FUTURE WORK

There are improvements that can be made to our model. The first of these is to use bounding information provided by admissible heuristics to provide good lower bounds to Constance. This can clearly aid forwards search by preventing many redundant layers being searched. It can also help in backwards search by lower-bounding the metric function.

One of the most interesting features of the CSP based approach to planning is the opportunity to extend the model with increasingly powerful constraints, pruning more of the search space rather than



(a) Blocksworld: A domain in which CPT dominates

(b) PipesNoTankage: A domain in which Constance dominates

(c) Airport: A domain in which SeP dominates

Figure 6. Time Performance. We show the time taken to solve instances from three domains. Each domain shows strong results for different planners.

leaving it to be explored. There remain many opportunities for extensions of our current model and we are currently investigating the idea of linking the CPT model, with its constraints, to ours, using channelling constraints to allow communication between the two models.

9 CONCLUSIONS

We have presented Constance, a constraint-based planner that uses analysis of substate graphs to discover and exploit composable action macros. Once discovered, these macros lead to the opportunity to exploit powerful dominance constraints. This opportunity makes the exploitation of these macros much more effective in this setting than has been the case for other work on macros. This is because other macro-based planners have typically faced a significant problem in controlling the explosion in branching factors caused by adding the macros alongside primitive actions, while also encouraging the planner to prefer macros where possible, often leading to poor quality plans. Constance achieves the successful exploitation of macros for cost-optimal planning. Constance improves on the performance of the current best constraint planners in many IPC domains.

Integrating ideas from heuristic search into constraint-based planners, and vice versa, provides many opportunities for future research. We also believe that integrating different constraint approaches can provide similar benefits.

Acknowledgements

We would like to thank Roman Bartak and Daniel Toropila for allowing us to use their planner SeP in our evaluation.

REFERENCES

[1] Roman Barták and Daniel Toropila, 'Revisiting constraint models for planning problems', in *Proc. 18th Int. Symp. on Foundations of Int. Systems*, pp. 582–591, (2009).
[2] Avrim Blum and Merrick L. Furst, 'Fast planning through planning graph analysis', *Artif. Intell.*, **90**(1-2), 281–300, (1997).
[3] Adi Botea, Markus Enzenberger, Martin Müller 0003, and Jonathan Schaeffer, 'Macro-ff: Improving ai planning with automatically learned macro-operators', *J. Artif. Intell. Res. (JAIR)*, **24**, 581–621, (2005).
[4] Christer Bäckström and Bernhard Nebel, 'Complexity Results for SAS+ Planning', *Computational Intelligence*, **11**, 625–656, (1995).
[5] Andrew Coles, Maria Fox, and Amanda Smith, 'Online identification of useful macro-actions for planning', in *Proc. 17th Int. Conf. Automated Planning and Scheduling*, pp. 97–104, (2007).
[6] Minh Binh Do and Subbarao Kambhampati, 'Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP', *Artif. Intell.*, **132**(2), 151–182, (2001).
[7] Stefan Edelkamp and Peter Kissmann, 'Optimal symbolic planning with action costs and preferences', in *Proc. 21st Int. Joint Conf. on AI*, pp. 1690–1695, (2009).
[8] M. Fox and D. Long, 'PDDL2.1: An extension of PDDL for expressing temporal planning domains', *Journal of AI Research*, **20**, 61–124, (2003).
[9] Maria Fox and Derek Long, 'Extending the exploitation of symmetries in planning', in *Proc. 6th Int. Conf. on AI Planning and Scheduling*, pp. 83–91, (2002).
[10] Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale, 'Data structures for generalised arc consistency for extensional constraints', in *Proc. 22nd Conf. AAAI*, pp. 191–197, (2007).
[11] Peter Gregory, Derek Long, and Maria Fox, 'A Meta-CSP Model for Optimal Planning', in *Proc. 7th Int. Symp. on Abstraction, Reformulation and Approximation*, pp. 200–214, (2007).
[12] Malte Helmert, 'The fast downward planning system', *J. Artif. Intell. Res. (JAIR)*, **26**, 191–246, (2006).
[13] Malte Helmert, 'Concise finite-domain representations for PDDL planning tasks', *Artif. Intell.*, **173**(5-6), 503–535, (2009).
[14] Malte Helmert and Carmel Domshlak, 'Landmarks, critical paths and abstractions: What's the difference anyway?', in *Proc. 19th Int. Conf. Aut. Planning and Scheduling*, (2009).
[15] Jörg Hoffmann, Julie Porteous, and Laura Sebastia, 'Ordered landmarks in planning', *J. Artif. Intell. Res. (JAIR)*, **22**, 215–278, (2004).
[16] John N. Hooker, 'A search-infer-and-relax framework for integrating solution methods', in *Proc. CP-AI-OR*, volume 3524 of *Lecture Notes in Computer Science*, pp. 243–257. Springer, (2005).
[17] Anders Jonsson, 'The role of macros in tractable planning over causal graphs', in *Proc. 20th Int. Joint Cont. AI*, pp. 1936–1941, (2007).
[18] Henry A. Kautz and Bart Selman, 'Planning as satisfiability', in *Proc. European Conf. AI*, pp. 359–363, (1992).
[19] Adriana Lopez and Fahiem Bacchus, 'Generalizing GraphPlan by Formulating Planning as a CSP', in *Proc. 18th Int. Joint Cont. on AI*, pp. 954–960, (2003).
[20] Silvia Richter, Malte Helmert, and Matthias Westphal, 'Landmarks revisited', in *Proc. 23rd Int. Conf. AAAI*, pp. 975–982, (2008).
[21] Vincent Vidal and Hector Geffner, 'Branching and pruning: An optimal temporal POCL planner based on constraint programming', *Artif. Intell.*, **170**(3), 298–335, (2006).