Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Transition Constraints for Parallel Planning

Nina Ghanbari Ghooshchi[1], Majid Namazi[1], M.A.Hakim Newton[2], Abdul Sattar[2]
[1]Computer Engineering Department, Urmia University, West Azarbaijan, Iran
[2]Institute for Integrated and Intelligent Systems (IIIS), Griffith University, Australia
{n.ghanbari, m.namazi}@urmia.ac.ir, {mahakim.newton, a.sattar}@grffith.edu.au

Abstract

We present a planner named Transition Constraints for Parallel Planning (TCPP). TCPP constructs a new constraint model from domain transition graphs (DTG) of a given planning problem. TCPP encodes the constraint model by using table constraints that allow *don't cares* or wild cards as cell values. TCPP uses Minion the constraint solver to solve the constraint model and returns the parallel plan. Empirical results exhibit the efficiency of our planning system over state-of-the-art constraint-based planners.

Introduction

Constraint-based planners translate a planning problem into a constraint satisfaction problem (CSP) and then apply CSP techniques. These planners thus allow to take advantage of enhanced propagation machineries and better pruning mechanisms in CSP to solve planning problems. However, in terms of performance, constraint-based planners do not yet match the state-of-the-art heuristic search planners. One key reason is that with the increase of plan length, the impact of propagation and pruning reduces greatly. Another main reason is that we still need a suitable constraint model that allows the best exploitation of the advanced CSP techniques. In this paper, we develop a new constraint model for planning problems to leverage the advancements in CSP.

Considerably little work has been done in solving planning problems by using CSP techniques. The first constraint models for planning problems were designed manually (van Beek and Chen 1999). Other models have been proposed in (Do and Kambhampati 2001), (Lopez and Bacchus 2003), and (Ghallab, Nau, and Traverso 2004). Most of these models translate the so called planning graph structures (Blum and Furst 1995) into CSPs with a view to generating parallel plans. Following the propositional nature of PDDL language (Ghallab et al. 1998), these planners mostly use Boolean variables and constraints with logical formulas.

Based on Fast Downward's (Helmert 2006) use of SAS+ formalism that is a member of the Simplified Action Structure family (Bäckström and Nebel 1995), constraints of a planning problem are encoded by table constraints (Barták and Toropila 2008). This has showed a great improvement in the efficiency. Inclusion of symmetry breaking, singleton consistency, lifting and nogood learning has further improved the performance (Barták and Toropila 2009a; 2009b). Recent constraint-based planners (Cesta and Fratini 2008; Verfaillie, Pralet, and Lemaître 2010; Barták 2011a) are based on time lines. The planner in (Gregory, Long, and Fox 2010) uses dominance constraints, and another one in (Judge and Long 2011) applies goal and variable/value heuristics and uses meta-CSP variables.

Inspired by the SAT-based planner SASE (Huang, Chen, and Zhang 2010) that exploits the structural information of SAS+ formalism, we develop a new constraint model for planning problems. Our planner named Transition Constraints for Parallel Planning (TCPP) extracts domain transition graphs (DTG) from the SAS+ representation and translates them into constraint models. We follow (Barták and Toropila 2008) to choose table constraints in our encoding model. Table constraints are efficient when the number of valid assignments is small with respect to the total number of assignments. However, we additionally and more importantly use *don't cares* or wild cards in table cells to allow a compact representation for many constraints that would otherwise need consideration of all possible combination of certain column values. To solve the encoded constraint problem, we use Minion the constraint solver (Gent, Jefferson, and Miguel 2006) that can handle table constraints with don't cares through algorithms for short support enabled general arc consistency propagation. This improves the efficiency of our planner over the state-of-the-art constraint-based planners on a set of standard benchmark domains.

The rest of the paper is organised as follows: next two sections discuss constraint-based planning and DTGs; the following section describes our new constraint model and our new constraint-based planner; the last two sections present our experimental results and conclusions.

Constraint-Based Planning

Besides the most successful family of forward chaining heuristic search planners, an alternative approach developed for planning is based on translation of the problem into a different formalism such as satisfiability (SAT) or CSP and then using respective solution techniques. One key issue in this approach is that SAT or CSP requires a static problem model whereas in planning the plan length is not known beforehand. A typical solution (Kautz and Selman 1992) to this

is to impose a fixed bound n on makespan (also called *horizon*) and translate the problem of finding a plan of makespan n as a SAT/CSP problem. If a plan is found, it is extracted; otherwise, a new horizon bound $n + 1$ is tried.

Most of the CSP-based planners produce sequential plans (Barták and Toropila 2008; Gregory, Long, and Fox 2010; Judge and Long 2011) and some other produce partially-ordered plans (Vidal 2004) and parallel plans (Barták 2011a; Do and Kambhampati 2001; Lopez and Bacchus 2003). Partial-order and parallel planners avoid examining symmetrical plans while the sequential planners do not. Symmetrical plans remain the same after changing action orderings. Exploring symmetrical plans is waste of time and also equivalence checking is time consuming.

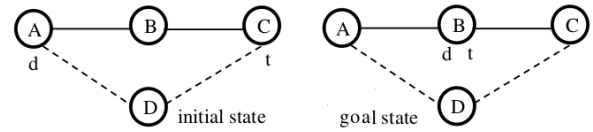
Constraint Models from Planning Graph: A dynamic CSP model in (Do and Kambhampati 2001) uses variables to represent the propositions at each level of the planning graph (Blum and Furst 1995) with the domains being actions supporting these propositions. Constraints used are action mutex, fact mutex, and subgoal activation constraints. A constraint model in (Lopez and Bacchus 2003) uses Boolean variables to represent the facts and actions at each level with constraints between variables being logical formulas that encode the initial state, goal state, preconditions and effects of actions and frame axioms. Another constraint model in (Ghallab, Nau, and Traverso 2004) uses Boolean variables.

Constraint Models from Multi-Valued Representation: Multi-valued representation of a planning problem leads to fewer variables with larger domains where domain filtering normally pays off. Further, the set of logical constraints from the original models can be encapsulated into combinatorial constraints with an extensionally defined set of admissible tuples. These constraints filter out more inconsistencies than the original logical constraints and the propagation loop can significantly reduce execution time. In (Barták 2011a), the idea of DTG is used but a finite state automata is considered for each state variable. A plan is an intersection of these automata. In this model, CSP variables are considered for states and actions in each time step and constraints are used in encoding the arcs of the automata and to synchronise the paths between different automata. In (Barták 2011b), a different encoding of DTG is proposed. Rather than encoding states and their changes via actions, the transitions between the actions are encoded directly and the state variables are omitted completely.

Domain Transition Graphs

The language used in the representation of a planning problem is an important factor in the efficiency of the formulation. The most common language used in describing planning problems is PDDL (Ghallab et al. 1998). Since this language is propositional, it is not suitable for encoding planning problems as CSPs. An alternative approach is SAS+ formalism (Bäckström and Nebel 1995) in which planning problems are represented by multi-valued state variables. This formalism became popular after being used in Fast Downward planner (Helmert 2006). By using this language, mutually exclusive predicates do not appear in the state description. Extracting structural information such as DTGs is also straightforward in this representation.

To describe this formalism, we will use a simple example from Driverlog Domain (see Figure 1). We restrict this domain to the transportation of drivers by trucks. We also omit packages. In this domain, we have some drivers and trucks. A driver can change its location by walking or by driving a truck. There are roads (solid lines) for driving trucks and footpaths (dotted lines) for drivers to walk. Relevant operators of this example are also listed in the figure.



board_truck(d, t, *l*)
preconds: t–loc = *l*, d–loc = *l*, t–occ = false
effects: d–loc = t, t–occ = true

disembark_truck(d, t, *l*)
preconds: t–loc = *l*, d–loc = t
effects: d–loc = *l*, t–occ = false

drive_truck (d, t, x, y)
preconds: t–loc =x, d–loc = t
effects: t–loc = y

l, x, y: locations
loc: location d: driver
occ: occupied t: truck

d–loc {A, B, C, D, t}
t–loc {A, B, C}
t–occ {false, true}

walk(d, x, y)
preconds: d–loc = x
effects: d–loc = y

Figure 1: A driverlog problem instance in SAS+.

In our example, we have four locations named: A, B, C, D; one driver d; and one truck t. The initial and goal states of the problem are depicted in Figure 1. To represent this example in SAS+ formalism, we need 2 state variables d-loc and t-loc for the locations of d and t respectively and a variable t-occ to denote whether the truck is occupied by a driver. Domains of these variables are shown in the figure.

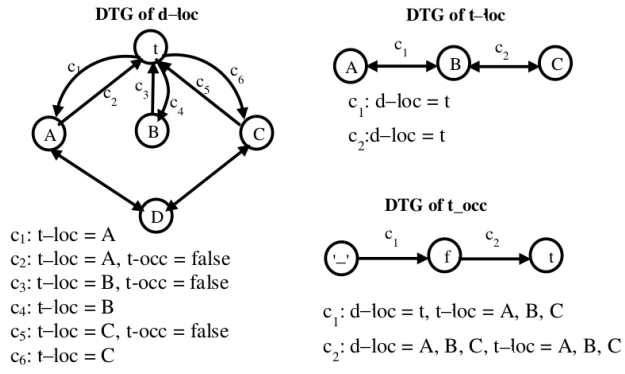


Figure 2: DTGs for the example in Figure 1.

From the SAS+ representation of a planning problem, we extract a DTG (Helmert 2006) for every state variable to show how these variables can change their values. For every value that a state variable can take, there is one vertex in its DTG. The edges describe how the values of the state variables change. If there is an action having a precondition

$v = d$ and an effect $v = d'$, we have an arc from the vertex with value d to the vertex with value d'. If an action has $v = d'$ as its effect but no precondition relating to variable v, then in the original DTG, there is an edge from every vertex to the vertex representing the value d'. In our version of DTGs, we simplify this by using a vertex with value '-' to denote a don't care value and draw an arc from this vertex to the vertex with value d'. Nevertheless, if any of the transitions is based on the value of other variable, that is if a condition such as $v' = d''$ is also included in the precondition of the action, this condition is mentioned on the transition edge. Actions responsible of changes are also included on the edges. DTGs for the example in Figure 1 are in Figure 2.

Our Constraint-Based Planner

We propose a new constraint model for parallel planning. In our model, DTGs extracted from the SAS+ representation of the planning problem are directly translated into table constraints. The main difference between this model and the state-of-the-art constraint-based planners is that we do not use any CSP-variables for actions taking place at each time. Our planner TCPP has three steps: Translation and Preprocessing, CSP Encoding and Solving, and Plan Decoding.

Translation & Preprocessing: An input PDDL problem is translated into a problem in SAS+ with multi-valued variables and corresponding instantiated actions (Helmert 2009). The SAS+ problem is preprocessed and DTGs are extracted by using our own extractor program similar to the one in the Fast-Downward planner (Helmert 2006). The DTGs used in our planner are slightly different than those used in Fast-Downward planner. For every action changing the value of a variable, we have an edge from the initial value of the variable to its final value while in Fast-Downward planner, not all of these edges are included.

CSP Encoding & Solving: In this step, a fixed bound n is imposed on the makespan. Then, the problem of finding a plan of makespan n is encoded as a CSP problem by using the DTGs. The encoded CSP problem is then solved by a CSP-solver. If a plan is not found for makespan n, the value of n is increased by one time unit and the process continues until a plan is found or a given time limit exceeds. To find the first length to start with, in each DTG, we look for the shortest path from the variable's value in the initial state of the problem to the value in its goal state and we choose the maximum of the shortest path lengths. Further details on our encoding and the solver being used are later in the paper.

Plan Decoding: The resultant plan is extracted from the solution returned by the CSP solver. To determine the actions at time τ, we look at the values of variables at time τ and $\tau + 1$ to see which one is changed. For every variable that changes its value, we look at its corresponding DTG to see which edges are responsible for this change. From the actions on these edges, the one having its preconditions satisfied at time τ and effects matching at time $\tau + 1$ is selected as a valid action. It is possible to have more than one action for each time τ. Thus our formulation allows parallel plans.

Our New Encoding

Procedure 1 in our planner translates the planning problem of makespan n to a CSP with the help of DTGs. Assume the planning problem has m state variables in its multi-valued representation. For every state variable v^i and for any time step $0 \leq \tau \leq n$, we have a CSP variable v^i_τ, which can take any value from v^i's domain. Therefore, we have $m(n + 1)$ CSP variables in our model. Procedure 1 uses four kinds of constraint: initial state constraints, goal state constraints, transition constraints, and negative constraints. Our contribution is a new encoding of the transition constraints.

Procedure 1 PlanningToCSP(n) // makespan n

//variables: v^i, v^j, values: p, q, actions: A, B
Foreach (v^i, p) in initial state, Add constraint eq(v^i_0, p)
Foreach (v^i, p) in goal state, Add constraint eq(v^i_n, p)
For timestep $\tau = 0$ **to** $n - 1$ **do**
  Foreach v^i, Add transition constraint tc(v^i_τ)
  Foreach pair $(A, B) : A$ is inconsistent with B
    Add a negative parallelism constraint
  Foreach pair $(v^i = p, v^j = q)$ in mutex groups
    Add a negative mutex table mt(v^i, p, v^j, q)

Initial and Goal State Constraints

Initial state constraints are simple constraints specifying the values of variables at time 0. Since the initial state is fully specified, we need *m* constraints for the values of the variables $v^0_0, v^1_0, \ldots, v^{m-1}_0$. The goal state is encoded in the same way with the difference that it is not fully specified.

Transition Constraints

For every DTG of a given problem, a transition constraint is added to our model. This constraint encodes all possible changes in the corresponding variable's value between time τ to $\tau + 1$. An edge in a DTG represents a possible way to change the value of the variable. This change can occur only if the conditions denoted on the edge are satisfied. The conditions specify the values that other variables should have so that the action that changes the value of this variable can occur. Because not all variables necessarily appear on one edge, the values of variables appearing on the other edges of the DTG are considered don't care for this change. Inspired by the use of tables constraints in (Barták and Toropila 2008) (although their model is different from ours), we have used this kind of *table constraints* to represent transitions.

Consider the DTGs in Figure 2 and their encoding as transition constraints in Figure 3. Suppose, we want to encode the DTG of d-loc to a transition constraint. The state variables involved in this DTG are d-loc itself, t-loc and t-occ. We need to consider the values of these 3 variables at time τ and $\tau + 1$. The table used for encoding has 6 columns d-loc$_\tau$, d-loc$_{\tau+1}$, t-loc$_\tau$, t-loc$_{\tau+1}$, t-occ$_\tau$, t-occ$_{\tau+1}$. After determining the columns, we have to find the rows of the table. Any DTG edge and hence the corresponding action can change the value of the d-loc and hence adds a table row.

The edge from A to t labeled with c_2 in the DTG denotes the board_truck. Due to this edge, d-loc can change from A to t at time τ if truck t is at location A and is not occupied.

We therefore have the 1st row in the table with values (A, t, A, A, false, true) for columns (d-loc$_\tau$, d-loc$_{\tau+1}$, t-loc$_\tau$, t-loc$_{\tau+1}$, t-occ$_\tau$, t-occ$_{\tau+1}$) respectively. The 2nd row corresponds to the edge from t to A with label c_1; notice that t-occ$_\tau$ is '-' because t-occ is not a precondition of the corresponding action disembark_truck. The 7th row in the table represents the walk action and thus the edge from C to D. The driver can change his location by walking between C and D. In this case, variables t-loc and t-occ can take any values and are thus considered don't care. Considering the three cases above, we can determine the table rows 1-10. Lastly, the driver can stay at the same location between successive times. We therefore have table rows 11-15 with d-loc having the same value at time τ and $\tau+1$ and the other variables having don't cares as their values.

TC for d-loc					
d-loc		t-loc		t-occ	
τ	$\tau+1$	τ	$\tau+1$	τ	$\tau+1$
A	t	A	A	F	T
t	A	A	A	-	F
B	t	B	B	F	T
t	B	B	B	-	F
C	t	C	C	F	T
t	C	C	C	-	F
C	D	-	-	-	-
D	C	-	-	-	-
A	D	-	-	-	-
D	A	-	-	-	-
A	A	-	-	-	-
B	B	-	-	-	-
C	C	-	-	-	-
D	D	-	-	-	-
t	t	-	-	-	-

TC for t-occ					
t-occ		d-loc		t-loc	
τ	$\tau+1$	τ	$\tau+1$	τ	$\tau+1$
F	T	A	t	A	A
F	T	B	t	B	B
F	T	C	t	C	C
-	F	t	A	A	A
-	F	t	B	B	B
-	F	t	C	C	C
F	F	-	-	-	-
T	T	-	-	-	-

TC for t-loc			
t-loc		d-loc	
τ	$\tau+1$	τ	$\tau+1$
A	B	t	t
B	A	t	t
B	C	t	t
C	B	t	t
A	A	-	-
B	B	-	-
C	C	-	-

Figure 3: Encoding domain transitions in DTGs in Figure 2 using table constraints. In the figure, Driver: d; Truck: t; Locations: A, B, C, D; Boolean: T, F; loc: location; occ: occupied; TC: transition constraint; '-': don't care; τ: time step.

Procedure 2 encodes a DTG into a table constraint. We first determine the columns in the table (see Step 1). For a state variable v^i with DTG G^i, the table T^i_τ at time τ has columns v^i_τ, $v^i_{\tau+1}$, v^j_τs, $v^j_{\tau+1}$s, where v^js are variables appearing on all edges of G^i. Each transition constraint is therefore a k-ary constraint where $k = 2(l+1)$ and l is the number of variables appearing on the edges of G^i. Next, we extract the rows of the table for each edge in the G^i (see Step 2). Suppose (p, q) is an edge in G^i and tr is the corresponding row to be added to the table. With respect to the edge, Row tr will have appropriate values in the relevant columns and don't cares '-' in the irrelevant columns. Because of edge (p, q), clearly, v^i_τ is p and $v^i_{\tau+1}$ is q; note p could be '-' in the G^i.

Procedure 2 tc(v^i_τ) *// transition-constraint*

G^i: DTG for v^i, T^i_τ: table for transitions of v^i_τ
$V = \{v^i\} \cup \{v^x : v^x$ appears on an edge in $G^i\}$
1. Foreach $v^k \in V$, T^i_τ has two columns v^k_τ and $v^k_{\tau+1}$
2. Foreach edge (p, q) in G^i *// p, q are values of v^i*
   tr: a new row in T^i_τ, each col has don't care '-'
   tr[v^i_τ] = p, tr[$v^i_{\tau+1}$] = q *// note p could be '-'*
   // Modify tr in the following ways
   Foreach condition $v^x = r$ on the edge (p, q)
      a_{pq}: the action associated with edge (p, q)
      If v^x only in precondition of a_{pq}
         tr[v^i_τ] = tr[$v^i_{\tau+1}$] = r
      ElseIf v^x both in precond and effect of a_{pq}
         Assume $v^x \leftarrow s$ in effect of a_{pq}
         tr[v^i_τ] = r and tr[$v^i_{\tau+1}$] = s
      ElseIf v^x only in effect of a_{pq} i.e. $r =$ '-'
         tr[$v^i_{\tau+1}$] = s where $v^x \leftarrow s$ in effect of a_{pq}
3. Foreach value p in domain of v^i
   tr: a new row in T^i_τ, each col has don't care '-'
   // Modify tr in the following ways
   tr[v^i_τ] = tr[$v^i_{\tau+1}$] = p

We now consider every condition $v^x = r$ on the edge and three cases to determine other relevant column values. Assume a_{pq} is the action corresponding to the edge (p, q).

1. v^x **only in a_{pq}'s precondition:** In this case, v^x is a prevailing condition for a_{pq} and remains the same at time τ and $\tau+1$ ensuring that during the execution of a_{pq}, v^x's value does not change by any other action.
2. v^x **in a_{pq}'s precondition and effect both:** Assuming $v^x \leftarrow s$ is in the effect, the values of v^x at time τ and $\tau+1$ in this case are trivially r and s respectively.
3. v^x **only in a_{pq}'s effect i.e. r is '-':** In this case, we need to specify the value of v^x only at time $\tau+1$, which we assume is s. At time τ, the value of v^x is don't care '-'.

Lastly, to allow v^i's value to be the same between successive time steps τ and $\tau+1$, we need one row for each value in the v^i's domain assuming don't care values for other variables in the row (see Step 3 in Procedure 2).

To summarise, we have *mn* transition constraints for a problem where *m* is the number of DTGs and n is the current makespan. Also, the number of rows in the table constraint for a DTG is $e + d$ where e is the number of edges in the DTG and d is the domain size of the variable.

Negative Constraints

While we need positive transition constraints to encode allowable changes, we also need negative constraints to prevent inconsistent changes from happening at the same time and between mutex groups in the SAS+ representation. We have two kinds of negative constraint in our encoding, mutex constraints and parallelism constraints.

Mutex Constraints: In the SAS+ representation of a planning problem, sometimes there are mutex groups containing a number of variable-value pairs that are mutually exclusive with each other. To ensure that no pair of these

occurs at the same time, we use mutex constraints. These constraints are 2-ary negative constraints. To summarise, for each mutex pair $(v^i = p, v^j = q)$ in mutex groups, we call procedure $\mathsf{mt}(v^i, p, v^j, q)$ to add a table with columns v^i_τ, v^j_τ and a row with column values p, q.

Parallelism Constraints: Not all instantiated actions can occur at the same time in a parallel plan. Two instantiated actions are inconsistent in the sense that they can not occur at the same time if there is a common state variable in their preconditions and effects and the values are conflicting. More precisely instantiated actions A and B are inconsistent if any of the four conditions hold (Barták 2011b):

1. Preconditions of A, B share a variable with conflicting values. $\exists(v^i = p) \in \text{pre}(A)\exists(v^j = q) \in \text{pre}(B)[v^i = v^j \wedge p \neq q]$
2. A variable in A's effect also appears in B's effect. $\exists(v^i \leftarrow p) \in \text{eff}(A)\exists(v^j \leftarrow q) \in \text{eff}(B)[v^i = v^j]$
3. A variable in A's effect appears in B's precondition. $\exists(v^i \leftarrow p) \in \text{eff}(A)\exists(v^j = q) \in \text{pre}(B) : [v^i = v^j]$
4. A variable in B's effect appears in A's precondition. $\exists(v^i \leftarrow p) \in \text{eff}(B)\exists(v^j = q) \in \text{pre}(A) : [v^i = v^j]$

Our model automatically prevents Case 1 from happening. It also automatically prevents Cases 2, 3 and 4 from happening but only when $p \neq q$. When $p = q$, we need negative constraints to ensure that instantiated actions A and B never happen at the same time. These negative constraints involve all state variables in the preconditions and effects of A and B and the tuples provide the non-allowed values of the corresponding CSP variables at time steps τ and $\tau + 1$. Construction of these negative tuples is similar to the construction of tuples for the edges in DTGs.

Theorem 1 *Our constraint encoding of a SAS+ problem is sound. Solving such a constraint problem for the lowest possible time horizon leads to an optimal parallel plan.*

We do not provide the proof of the above theorem. However, the proof can be easily constructed by finding the correspondence of the transition constraints and the negative constraints with the semantics of the planning language and the validity conditions of the plans. The optimality notion comes from the GraphPlan algorithm (Blum and Furst 1995).

Solving Our CSP Model

To solve the resulting CSP model, we use the open source constraint solver Minion with singleton arc consistency and dom/wdeg (conflict) for variable ordering. To the best of our knowledge, Minion is the only solver that implements SHORTSTR2 propagation (Jefferson and Nightingale 2013). SHORTSTR2 is an efficient and effective general purpose propagation algorithm for exploiting *short supports*. A *support* in a constraint for a domain value of a variable is a justification that the value may still form part of an assignment that satisfies the constraint. Some constraints can be satisfied by a *short assignment* that is assigning only a few of their variables. After the short assignment, the constraint doesn't care about the values of the rest. A short assignment that satisfies the constraint is called a *short support*. Because of don't-care values in our table constraints, we use Minion with SHORTSTR2. We selected singleton arc consistency and dom/wdeg in Minion by testing various options. Without arc consistency, the CPU time increases in most domains. Also, dom/wdeg was apparently found to be the best option when tried on a number of problems. To use Generalised Arc Consistency (GAC) propagation implemented in all solvers, we need to expand all the don't-care values in our constraints. As a result, the number of valid assignments increases and the efficiency of GAC propagation severely decreases. In our encoding, the number of valid assignments is fewer than the number of all possible assignments. Exploiting SHORTSTR2 which works on constraints with don't-care values, our planner thus becomes very efficient.

Experimental Results

We ran experiments on a PC with Intel 3.5GHz CPU, 8GB memory limit and 30 minute timeout. Our time measurement starts from the input PDDL to the output plan. We tested our planner on 12 domains from past International Planning Competition (IPC) STRIPS versions. We compare our planner with two constraint-based planners Constance (Gregory, Long, and Fox 2010) and PaP2 (Barták 2011b). We choose these planners because they represent the current state of the art of constraint based planning. Constance is a sequential planner that performs better than CPT (Vidal and Geffner 2006) and SeP (Barták and Toropila 2008). SeP performs better than GP-CSP (Do and Kambhampati 2001) and CSP-plan (Lopez and Bacchus 2003). PaP2 is a parallel planner that is faster than SeP and PaP1 (Barták 2011a). To check how TCPP performs compared to the state-of-the-art heuristic search planners, we also run Fast Downward with option –search astar(lmcut()) and denote it by FDLMC.

Planning Domains	Instance Count	Problem Instances	Instances Solved			
			Const	PaP2	TCPP	FDLMC
Airport	15	1-15	0	14	14	**15**
Blocks	35	4.0-17.0	13	13	**32**	28
Depot	10	1-10	2	3	**7**	6
Driverlog	15	1-15	6	13	13	**14**
Grid	05	0-5	1	1	**2**	**2**
Gripper	10	0-10	1	2	2	**7**
Logistics00	28	4.0-15.1	6	15	**24**	20
Logistics98	10	0-10	0	**5**	04	2
Miconic	35	1.0-7.4	27	34	27	**35**
Rovers	20	1-20	4	11	**18**	9
Tpp	15	1-15	4	8	**10**	7
Zenotravel	15	1-15	0	12	12	**13**
All domains	213		64	131	165	158

Figure 4: Numbers of problem instances solved by planners when ran with 30-min-timeout on the same computer. Boldfaced values denote the winner planner in the domain.

In Figure 4, we have reported the number of tested and solved problem instances in each of the 12 IPC domains. We observe that in Blocks, Depot, Grid, Logistics00, Rovers, and TPP domains, TCPP is a clear winner over PaP2 while PaP2 is a winner on Miconic and Logistics98. Planners PaP2 and TCPP perform similar in other 4 domains. Both TCPP

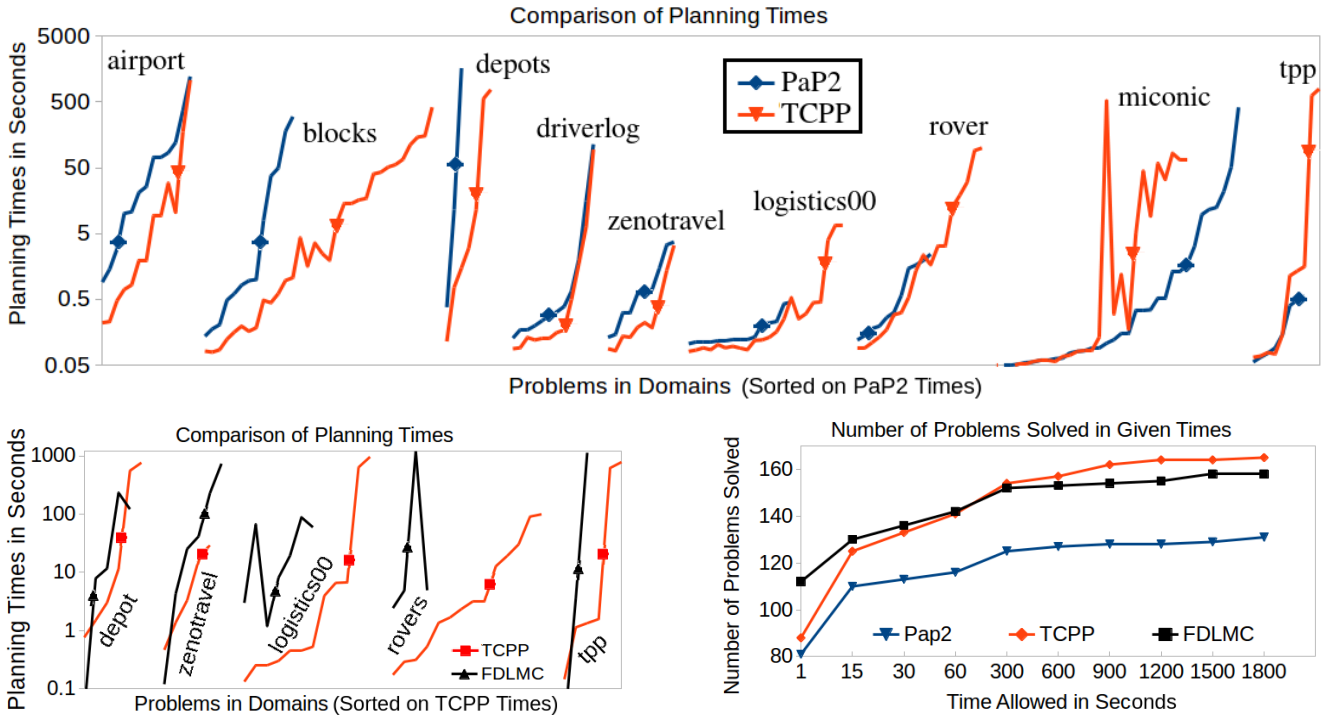


Figure 5: *Top:* Time performance by PaP2 and TCPP on 9 domains where TCPP solved at least 5 instances. *Bottom-Left:* Time performance of TCPP and FDLMC on 5 domains (comparatively challenging for FDLMC) with instances requiring at least 0.1 second by one of the planners. *Bottom-Right:* Total number of problems solved over all 12 domains when less-than 30 minute time-cutoffs are used for each planner on each problem instance in each domain.

and PaP2 solves much more problems than Constance in most domains. TCPP also wins over FDLMC in Blocks, Logistics00, Rovers and TPP. To summarise, 213 problems have been attempted in total in the 12 domains. Constance, PaP2, and FDLMC can respectively solve 64 (i.e. 30%), 131 (i.e. 61.5%), and 158 (i.e. 74.2%) of them while TCPP can solve 165 (i.e 77.5%). Clearly, TCPP can solve 16% more problems than PaP2 and 3.3% more than FDLMC.

Rather than showing the performance at the end of 30-minute timeout, Figure 5 (Bottom-Right) depicts the total number of problems solved if certain less-than-30-minute time-cutoffs are used for each planner on each problem in each domain. All domains considered together, TCPP is significantly better in solving more problems than PaP2 regardless of the exact time cutoff (less than 30-minutes) being used. Interestingly, FDLMC can solve more problems than TCPP with smaller timeouts, but after few minutes TCPP-starts solving more problems and wins at the end.

Figure 5 (Top) shows the planners' performance on problem instances in 9 domains where TCPP could solve at least 5 instances. For space constraints, we do not show the other 3 domains where performance is similar for both planners. Note the times on the y-axis are in logarithmic scale. Overall, TCPP solves more problems than PaP2 in most domains and solves problems faster in most domains as well. The differences in performance is huge in domains such as airport, blocks, and depots. In driverlog and zenotravel, TCPP is consistently better. In logistics00, rover, and tpp, the gap is narrow but more problems have been solved. TCPP performed worse than PaP2 in Miconic where TCPP solved fewer instances and was significantly slower as well.

In Figure 5 (Bottom-Left), we compare FDLMC and TCPP in 5 domains on problem instances where one of the planners take at least 0.1 second; TCPP clearly outperforms FDLMC. Just for example, we list few specific instances where TCPP won over FDLMC, depot-4(11.44sec vs 233.34sec), logistics00-11-0(0.46sec vs 88.36sec), rovers-8(19.18sec vs 1468.14sec), tpp-7(1.35sec vs 1137.96sec), zenotravel-12(29.16sec vs 230.48sec). However, FDLMC solves all airport and miconic problems in negligible time. Also, in other domains on problem instances that FDLMC can solve, it solves in much less time than TCPP.

Lastly, we report the maximum number of columns in the encoded tables as follows: depot 108, blocks 70, tpp 20, driverlog 18, zenotravel 12, logistics00 16, logistics98 76, grid 42, gripper 48, miconic 6, airport 440, and rovers 50.

Conclusions

We have presented a new constraint model for parallel planning. We translate the domain transition graphs of a planning problem into table constraints with don't cares and use advanced constraint solving techniques. Our planner thus outperforms existing constraint-based planners in terms of solution time and the number of problem instances solved.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–656.

Barták, R., and Toropila, D. 2008. Reformulating constraint models for classical planning. In *Proceedings of the International FLAIRS Conference*, 525–530.

Barták, R., and Toropila, D. 2009a. Enhancing constraint models for planning problems. In *Proceedings of the International FLAIRS Conference*.

Barták, R., and Toropila, D. 2009b. Revisiting constraint models for planning problems. In *International Symposium on Methodologies for Intelligent Systems*, 582–591.

Barták, R. 2011a. A novel constraint model for parallel planning. In *Proceedings of the International FLAIRS Conference*.

Barták, R. 2011b. On constraint models for parallel planning: The novel transition scheme. In *Proceedings of the Eleventh Scandinavian Conference on Artificial Intelligence*, 50–59.

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1):1636–1642.

Cesta, A., and Fratini, S. 2008. The timeline representation framework as a planning and scheduling software development environment. In *Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group*.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning-graph by compiling it into CSP. *Artificial Intelligence* 132:151–182.

Gent, I. P.; Jefferson, C.; and Miguel, I. 2006. MINION: a fast, scalable, constraint solver. In *Proceedings of the European Conference on Artificial Intelligence*.

Ghallab, M.; Knoblock, C.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D. E.; Sun, Y.; and Weld, D. 1998. PDDL: the planning domain definition language. In *Technical Report, Yale University*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning - Theory and Practice*. Elsevier.

Gregory, P.; Long, D.; and Fox, M. 2010. Constraint based planning with composable substate graphs. In *Proceedings of the European conference on Artificial Intelligence*, 453–458.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5-6):503–535.

Huang, R.; Chen, Y.; and Zhang, W. 2010. A novel transition based encoding scheme for planning as satisfiability. In *Proceedings of the Twenty-Fourth National Conference on Artificial intelligence (AAAI)*.

Jefferson, C., and Nightingale, P. 2013. Extending simple tabular reduction with short supports. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*.

Judge, M., and Long, D. 2011. Heuristically guided constraint satisfaction for planning. In *Proceedings of the 29th Workshop of the UK Planning and Scheduling Special Interest Group*.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the European Conference on Artificial Intelligence*, 359–363.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a CSP. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 954–960. Morgan Kaufmann Publishers.

van Beek, P., and Chen, X. 1999. CPlan: A constraint programming approach to planning. In *Proceedings of the Sixteenth National Conference on Artificial intelligence*, AAAI '99/IAAI '99, 585–590.

Verfaillie, G.; Pralet, C.; and Lemaître, M. 2010. How to model planning and scheduling problems using constraint networks on timelines. *Knowledge Engineering Review* 25(3):319–336.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programmming. *Artificial Intelligence* 170(3):298335.

Vidal, V. 2004. Branching and pruning: An optimal temporal POCL planner baed on constraint programming. In *Artificial Intelligence*, 570–577.