

Constraint Partitioning for Solving Planning Problems with Trajectory Constraints and Goal Preferences

Content Areas: Planning and Scheduling, Search, Constraint Satisfaction

Paper ID: #1259

Abstract

The PDDL3 specifications include soft goals and trajectory constraints for distinguishing high-quality plans among the many feasible plans in a solution space. To reduce the complexity of solving a large PDDL3 planning problem, constraint partitioning can be used to decompose its constraints into subproblems of exponentially lower complexity. However, constraint locality due to soft goals and trajectory constraints cannot be effectively exploited by existing subgoal-partitioning techniques developed for solving PDDL2.2 problems. In this paper, we present an improved partition-and-resolve strategy for supporting the new features in PDDL3. We evaluate techniques for resolving violated constraints, optimizing goal preferences, and achieving subgoals in a multi-valued representation. Empirical results on the 5th International Planning Competition (IPC5) benchmarks show that our approach is effective and significantly outperforms other competing planners.

1 Introduction

As plan quality is a major issue in many planning problems, traditional quality criteria in PDDL2.2 planning, like plan length, are inadequate. Inspired by real applications, soft goals and trajectory constraints have been introduced in the PDDL3 specifications [Gerevini and Long, 2005].

Soft goals, in conjunction with penalties, can be used for modeling goal preferences. For instance, one may prefer storing `crate1` in `depot1` to storing `crate1` in `depot2`, since the latter leads to a higher violation cost in the plan metric value. Unlike traditional planning that requires a complete satisfaction of all the goals, planning with soft goals entails the selection of an appropriate subset of the soft goals when it is infeasible to achieve all of them. Hence, an exhaustive search may be needed in order to identify the best subset.

Trajectory constraints, on the other hand, are hard or soft constraints over intermediate states during plan execution. They are used to express temporal logic in planning and to distinguish the many feasible paths to a goal state. Informally, for a plan trajectory $\pi = \langle (S_0, t_0), (S_1, t_1), \dots, (S_n, t_n) \rangle$,

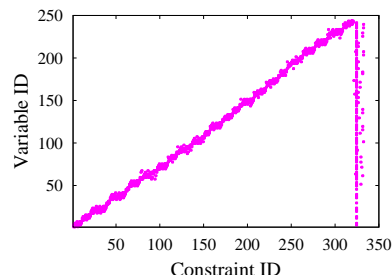
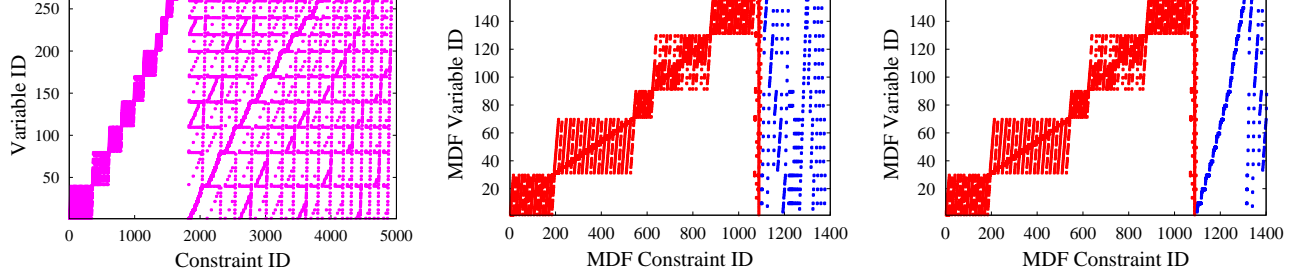


Figure 1: The constraint-variable structure depicting a strong constraint locality when the constraints of the IPC5 *Rovers-Propositional-30* instance is clustered by its subgoals.

where S_i is the intermediate state at time t_i , a trajectory constraint can be defined by a logical formula, respectively, at the end of, to be always true in, some time in, some time before t of, or at most once in the trajectory. A trajectory constraint can also defined to enforce one formula before another, or to enforce two formulas within time t in the trajectory. Clearly, trajectory constraints pose more challenges on planning, even when the number of actions or facts is not increased.

In view of the new features in PDDL3 problems, we study in this paper a partition-and-resolve strategy for solving these problems efficiently. We partition a large problem by its constraints into subproblems, solve the subproblems individually by an existing but modified planner, and resolve those inconsistent global constraints across the subproblems. Constraint partitioning has been demonstrated to be a very promising technique for solving large planning problems because each subproblem has only a fraction of the original constraints and is a significant relaxation of the original problem. As a result, each subproblem is of exponentially lower complexity as compared to that of the original problem. The technique, however, leads to global constraints across the subproblems, which can be effectively resolved using the extended saddle-point condition developed in [Wah and Chen, 2006].

The original partition-and-resolve strategy in [Chen *et al.*, 2006] (hereafter called `SGPLAN4`) was developed for solving PDDL2.2 problems whose goals are a conjunctive list of facts. Constraint partitioning in `SGPLAN4` exploits the strong locality of mutex constraints in many IPC4 benchmarks [Edelkamp and Hoffmann, 2004] when the constraints are clustered by their subgoals. Here, *constraint locality* is measured by the fractions of constraints that are global constraints, which span across multiple subproblems after the



a) Naive clustering by soft goals b) Clustering by hard constraints in MDF c) Clustering by hard/soft constraints in MDF

Figure 2: The constraint-variable distributions (after some rearrangement) of the *TPP-QualitativePreferences-5* instance that show the different degrees of constraint locality using three ways of clustering.

problem has been partitioned. Figure 1 shows a strong constraint locality after clustering the constraints of the *IPC5 Rovers-Propositional-30* instance [Gerevini *et al.*, 2006] by its subgoals. Because the goal is indeed a conjunctive list of facts, partitioning by subgoals works well for this instance.

Subgoal partitioning, however, is not always effective for PDDL3 domains. PDDL3 problems can have implicit constraints on their final state, using logical expressions like (imply (stored goods11 level1) (stored goods18 level2)). Moreover, the presence of soft goals and trajectory constraints may lead to different constraint locality from that observed in PDDL2.2 domains. Figure 2a depicts the constraint-variable distribution after clustering the constraints of the *TPP-QualitativePreferences-5* instance by its soft goals. Because trajectory constraints are not considered in clustering, the result leads to poor constraint locality.

In clustering constraints of IPC5 instances, we observe that the propositional representation of binary facts in *SGPlan₄* usually obscures many mutual exclusions and requires superfluous facts. For example, the location of truck1 in the *TPP* domain is represented by several binary facts, like $\text{at}(\text{truck1}, \text{market1}), \dots, \text{at}(\text{truck1}, \text{market8})$, with some implicit constraints for enforcing their consistency. To have a more compact representation of these constraints, we use a *multi-valued domain formulation* (MDF) in this paper based on the SAS+ formalism [Bäckström and Nebel, 1995]. MDF has been used in several planners, including Fast Downward and the IP planner. This compact representation makes implicit mutexes explicit and reduces the number of global constraints across the subproblems. For instance, the location of truck1 in *TPP* can now be represented by one variable with multiple values.

Based on the multi-valued variables, we can derive the possible transitions among their values. For instance, $\text{market1} \leftrightarrow \text{market2}$ represents the bidirectional connection between two markets. We can also derive their causal dependencies, where variable a has a causal dependency on variable b if a transition in a has a state requirement on b . For example, in order for truck1 to move package1 from location1 to location2, it has to be at location1. In this case, the transition for package1 has a causal dependency on the location of truck1.

Our main contributions in this paper are the design of a new strategy for clustering and for partitioning the constraints of PDDL3 planning problems and the demonstration of its success in a prototype planner. Based on the constraint locality observed, we partition mutexes and hard and soft trajectory

constraints of a planning problem into loosely related subproblems by some multi-valued state variables. We study various design options in partitioning and demonstrate their improvements in constraint locality. To handle the new features in PDDL3, we develop new techniques, both at the global and the subproblem levels, for optimizing goal preferences and for resolving trajectory and mutex constraints.

2 Partitioning Strategy

In this section, we first illustrate the constraint locality of a PDDL3 problem when it is partitioned by its trajectory constraints and soft goals. We then present our partitioning algorithm and the results on some IPC5 benchmarks.

2.1 Constraint Locality by Trajectory Constraints and Soft Goals

As is shown in Figure 2a, clustering the constraints of a PDDL3 problem by its subgoals does not lead to high constraint locality. Hence, our goal in this section is to identify new problem-independent attributes that allow PDDL3 problems to be partitioned with better constraint locality.

We have formulated a PDDL3 planning problem in such a way that multi-valued variables represent a solution plan that satisfies mutual-exclusion, trajectory, and goal-state constraints. We have found that constraint locality can be improved when some state variables in goal-state constraints (called *guidance variables*) are used for clustering the constraints of a problem into subproblems. Since the goal-state representation is also a conjunction of facts, the framework in *SGPlan₄* can be extended to solve the partitioned problem.

Figure 2b illustrates the constraint locality when we cluster the mutex constraints (red points) in *TPP-QualitativePreferences-5* by the five guidance variables that represent the stored quantity of the five products. Although constraint locality is greatly improved over that in Figure 2a, the result is still unsatisfactory because there are a number of soft constraints (blue points) that are not localized.

To address this issue, we further cluster the soft constraints by the same guidance variables. Figure 2c shows that all the constraints are now clustered well, leading to a good constraint locality. Note that we may have constraints that involve a limited number of guidance variables yet exhibit strong constraint locality. In the next section, we study the granularity of partitioning in order to further reduce the number of global constraints.

Table 1: Trade-offs on the number of partitions for the *Trucks-TimeConstraints-20* instance.

Partitioning Strategy	No Partitioning	Bottleneck	Guidance
# partitions	1	4	22
# global constraints	0	20	573
avg. # local const. per subproblem	21274	1404	230.1
time/subproblem	>1800 sec.	2.04 sec.	0.16 sec.

Bottleneck: # partitions = min(# bottleneck var., # guidance var.)

Guidance: # partitions = # of guidance variables

Subgoal: subgoal partitioning

2.2 Proposed Partitioning Algorithm

Based on our observation that constraint locality is associated with some guidance variables, we present in this section our algorithm for identifying the guidance variables, selecting a suitable number of partitions, and clustering the constraints by these variables. As the algorithm requires all the constraints to be expressed in MDF, we have implemented a pre-processing engine by following the techniques in [Helmert, 2004] for translating a PDDL representation into MDF.

In addition to MDF analysis, we also identify and remove *accompanying state variables* in order to make the representation more compact. These are state variables whose values can be inferred from the values of other state variables. For example, $\text{on}(\text{crate1}, \text{depot0-1-1})$ implies $\text{in}(\text{crate1}, \text{depot0})$, given that $\text{in}(\text{depot0-1-1}, \text{depot0})$ is true. This identification is possible because action *add* on $(\text{crate1}, \text{depot0-1-1})$ must also *add* $\text{in}(\text{crate1}, \text{depot0})$, whereas action *delete* on $(\text{crate1}, \text{depot0})$ must also *delete* on $(\text{crate1}, \text{depot0-1-1})$. For each candidate state variable, we test if all its values have the above relationship with the values of other state variables.

After eliminating the accompanying state variables, we define the partitioning guidance variables to be the set of variables in the goal-state constraints. Once the values of these variables have been determined by satisfying all the hard goal constraints and by minimizing the penalties of the soft goal constraints to some extent, we define the goal of a subproblem to be a partial assignment of the set of guidance variables. We also specify the maximum number of partitions to be the number of guidance variables.

The number of partitions is an important parameter of our partitioning algorithm. Although it is desirable to enumerate different numbers of partitions and to study their trade-offs, it is not the best approach here because there are some problem-dependent features that can be utilized. We employ a logical choice through the identification of bottleneck variables. In various planning domains, some objects are the sources of mutexes because actions want to concurrently access a limited number of these objects. These can be treated as bottleneck resources of parallelism. Specifically, we search for a group of state variables of the same type, and any changes of state variables must depend on these *bottleneck variables*. An example is the locations of trucks (*resp.* hoists) in the *TPP* (*resp.* *Storage*) domain. Our strategy is to set the number of partitions to be the smaller of the number of guidance variables and the number of bottleneck variables.

Table 1 shows the trade-offs in granularity on the IPC5 *Trucks-TimeConstraints-20* instance. When the instance is

Table 2: Average fraction of constraints that are active global constraints initially across all the instances of each IPC5 *QualitativePreferences* domain under three partitioning strategies. (See keys in Table 1.)

Domain	Bottleneck	Guidance	Subgoal
<i>TPP</i>	0.0153	0.0733	0.2892
<i>OpenStacks</i>	0.0000	0.1728	0.1834
<i>Trucks</i>	0.0000	0.0749	0.1623
<i>Storage</i>	0.0007	0.0032	0.0109
<i>Rovers</i>	0.0227	0.0464	0.0488

partitioned with respect to the bottleneck variables, the result confirms our claim in Section 1 that the time for solving a subproblem decreases by three orders of magnitude, while the number of partitions (and the number of global constraints) is increased by a small number. However, as the number of partitions is increased further (when partitioning is done with respect to the guidance variables), there is diminishing reduction in the time to solve a subproblem, while the number of global constraints is dramatically increased. In this case, partitioning is best done with respect to the bottleneck variables.

Finally, we formulate the problem of clustering the constraints by the guidance variables into a graph partitioning problem. We define a node in the graph to be a guidance variable, and an edge between two nodes when there is a constraint on these variables. We do not include those constraints that are automatically satisfied by the values of the guidance variables. For example, $(\text{imply}(\text{stored goods11 level1})(\text{stored goods18 level2}))$ is true if we have determined the value of $\text{stored}(\text{goods11})$ to be *level1*, and the value of $\text{stored}(\text{goods18})$ to be *level2*. For preferences on soft constraints, we set the weight on an edge to be the violation cost of the corresponding soft constraint. Last, we apply any competent graph partitioning software, such as METIS (<http://glaros.dtc.umn.edu/gkhome/views/metis/>), to cluster the constraints into groups that are related by a minimal number of global constraints.

2.3 Results on Some IPC5 Benchmarks

In this section, we report the evaluation results of our proposed partitioning strategies on some IPC5 benchmarks. We have chosen the *QualitativePreferences* track whose instances have both soft goals and trajectory constraints. Table 2 shows the average fraction of all constraints that are active global constraints initially for three partitioning strategies. Some entries in the table are zeroes because no partitioning was done for those domain-strategy combinations.

The result shows that partitioning based on guidance variables leads to better constraint locality than that of subgoal partitioning. Further, our strategy for controlling granularity can avoid exorbitant resolution overheads when the locality is weak and when a subproblem is too complicated to be solved without partitioning. We address this issue in Section 3.2 using subproblem-level decomposition techniques. Note that all three partitioning strategies work well for the *Storage* domain because its constraint locality is very strong. However, since the instances in this domain have a huge number of constraints, even a small fraction of violated global constraints would need to be resolved in many iterations.

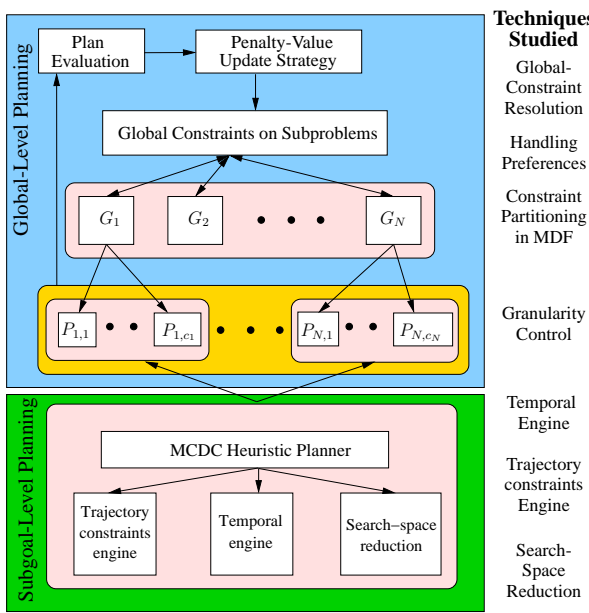


Figure 3: Architecture of SGPlan₅.

3 Architecture of SGPlan₅

SGPlan₅ partitions a planning problem into subproblems, each with a goal state, and finds a feasible plan for each (Figure 3). In the global level, it partitions the problem by its guidance variables and resolves any violated global constraints. In the local level, it calls a modified Metric-FF planner for solving each subproblem, using the violated global constraints and the global preferences as biases.

3.1 Global-Level Search

SGPlan₅ solves a planning problem formulated in mixed space by a nonlinear objective and one or more constraints. Using a penalty function of the objective and the transformed constraint functions weighted by their penalties, it looks for extended saddle points (ESPs) of the function [Wah and Chen, 2006]. Here, an ESP is a local minimum of the penalty function with respect to the original variables and a local maximum with respect to the penalties for all penalties larger than some thresholds. The algorithm is based on the ESP condition (ESPC), which states the one-to-one correspondence between an ESP and a feasible local minimum of the constrained model.

The planner first decides on the values of the partitioning guidance state vector x before partitioning the constraints. The metric value for each assignment x' of x is estimated by solving the following relaxed problem:

$$\begin{aligned}
 \min_{\pi} \quad & J(x', \pi) = J_x(x') + J_{\pi}(\pi) \\
 \text{subject to} \quad & H(\pi) = 0, \quad R(x') = 0 \\
 \text{and} \quad & S_n \models x',
 \end{aligned} \tag{1}$$

where $\pi = \langle (S_0, t_0), \dots, (S_n, t_n) \rangle$ defines the plan trajectory, $H(\pi)$ is the set of hard constraints, and $R(x)$ is the reachability test. The metric function $J(x', \pi)$ consists of J_x whose value can be wholly inferred from x' , and J_{π} whose

value is calculated by evaluating the plan trajectory. Without additional restriction on π , we prune cases in which the violation of H can be deduced from x' .

We find the best x' by a heuristic search with backtracking. Since $J_{\pi}(\pi)$ is unknown until planning ends, we use $\tilde{J}(x', \pi)$, the plan metric value of the relaxed plan for x' , to estimate $J(x', \pi)$. This relaxed plan is provided by the basic planner and is used for reachability tests as well. If $J_{\pi}(\pi)$ is zero, then the above search reduces to an enumeration of all reachable assignments of x . In that case, the solution is optimal when the best x' for (1) is found. In general, our strategy does not guarantee plan optimality because the basic planner does not ensure the optimality of $J_{\pi}(\pi)$.

To reduce the complexity of backtracking or enumeration, we decompose the search by utilizing the locality of goal-state constraints. This is possible because almost all soft goals involve only a small number of state variables.

The handling and enumeration of general trajectory constraints is difficult because it involves a number of intermediate states rather than one final state. Further, due to the large feasible space, inconsistencies between constraints are difficult to detect. Our approach is to divide the trajectory constraints into local and global constraints. Hard local constraints are enforced by the basic planner, whereas global constraints are resolved by the penalty method. We follow a similar procedure for resolving mutual exclusions in handling global-level trajectory constraints. Since soft constraints are not always satisfiable, we compute the plan metric at the end of each iteration and record the incumbent. Note that the penalty values used for resolving constraint violations differ from preference weights used for computing the metric value.

3.2 Subproblem-Level Search

Our basic planner follows Metric-FF’s heuristic algorithm [Hoffmann, 2003], but employs new heuristic functions based on MDF to handle the new features in PDDL3. Using MDF, we have implemented a new heuristic by exploring the value-transition graph of each variable and the causal dependencies between the transition graphs. The idea is inspired by and similar to that in Fast Downward [Helmert, 2004].

Our *minimum causal dependency-cost* (MCDC) heuristic employs a complete recursive depth-first search for generating a heuristic plan with the minimum cost without pruning the causal graphs. In contrast, the Fast Downward heuristic is incomplete since it performs strongly-connected-component analysis and removes nodes with low connectivity. Although MCDC provides a more accurate heuristic plan, its efficiency cannot compete with Metric-FF’s “ignoring-deletelists” heuristic function. We use MCDC as a better dead-end detector and metric-value estimator.

We have implemented the parser and the preprocessor for supporting PDDL3 domains. To synchronize the checks of constraints and the evaluation of violation costs, we encode each constraint (*resp.* preference value) with an artificial predicate (*resp.* function). All constraints are grounded, and quantifiers are compiled away. During state transitions, the values of these artificial fluents are derived from the existing trajectory using axioms. Having the above encoding and

modification allows us to perform a breadth-first search (BFS) in satisfying constraints or in optimizing preferences.

To improve the efficiency of BFS, we incorporate new heuristics for constraint satisfaction with Metric-FF’s heuristic function. Our solution is to add artificial facts of local constraints into the goal state of a subproblem, and compute the combined heuristic value as the sum of the heuristic value for the constraints and that for the subgoal. Since it is not meaningful to sum heuristic values for hard and soft constraints, we penalize soft- or global-constraint violations by iteratively increasing their weights.

Before solving a partitioned subproblem, we can often eliminate many irrelevant actions in its search space. We identify relevant actions by traversing the causal graphs in MDF and by ignoring those actions that are not useful for achieving the current subgoal variables. We also prioritize actions that do not cause an inconsistent assignment of MDF variables. This is done by first finding the set of bottleneck variables and by applying the helpful-action idea in FF [Hoffmann and Nebel, 2001] to defer those actions that concurrently change the value of the bottleneck variables.

With the subproblems generated, it is possible that a subproblem is too complicated to be solved, especially when the proposed partitioning strategy chooses a small number of partitions. Reducing the grain size further would not be effective due to the complexity of resolution. For instance, all the subproblems in the IPC5 *OpenStacks-Propositional* domain are trivial to solve, but the major challenge is to resolve the global constraints. In case that the basic planner fails to find a feasible subplan, we use incremental planning [Hsu *et al.*, 2005] or landmark analysis [Hoffmann *et al.*, 2004] to further decompose a subproblem into more manageable units.

4 Experimental Results

We first evaluate the performance of our proposed partitioning strategy, which selects the number of partitions to be the minimum of the number of bottleneck variables and the number of guidance variables. We test our proposed strategy on a number of large IPC5 instances. We do not select instances from domains with preferences, since most of them can be readily solved by relaxing all their soft constraints.

Table 3 shows the slowdowns of solving each instance using some fixed numbers of partitions with respect to our proposed partitioning strategy. It shows that our proposed strategy is usually the best, except in solving the *OpenStacks-MetricTime* domain. Although *OpenStacks-MetricTime* has a similar problem structure as that of *OpenStacks-Propositional*, our strategy fails to detect its weak locality because it does not find any bottleneck variables. Moreover, the benefit of partitioning in this domain is not tangible because its instances are rather easy.

Table 4 summarizes the statistics of IPC5 instances solved by SGPLan₅ and shows that it can solve about 90% of all the instances. Most of the unsolved instances are in the *MetricTimeConstraint* track and the *PipesWorld* domain. We have found that our heuristics are not effective for handling trajectory constraints with one formula before another, either quantitatively or qualitatively. Hence, one promising topic to be

Table 3: Slowdowns of solving some large IPC5 instances by some fixed numbers of partitions with respect to our proposed partitioning strategy.

IPC5 Instance	Proposed Strategy		# of Partitions				
	(g, b)	$\min(g, b)$	1	2	4	8	g
<i>TPP-PR-30</i>	(20,8)	8	—	—	4.98	1.00	1.72
<i>TPP-MT-40</i>	(25,10)	10	—	16.35	2.92	2.01	1.29
<i>TPP-MTC-22</i>	(4,2)	2	—	1.00	—	—	—
<i>OpenStacks-PR-30</i>	(100,1)	1	1.00	3.34	4.09	3.34	5.01
<i>OpenStacks-MT-30</i>	(50,N/A)	50	0.76	0.76	0.76	0.76	1.00
<i>Storage-PR-30</i>	(20,5)	5	—	—	45.66	7.64	0.72
<i>Storage-TI-30</i>	(20,5)	5	—	—	40.68	6.15	9.23
<i>Storage-TC-9</i>	(9,5)	5	19.23	3.45	1.98	1.01	0.98
<i>Trucks-PR-20</i>	(20,1)	1	1.00	7.82	10.45	—	—
<i>Trucks-TI-30</i>	(32,5)	5	3.27	0.71	0.57	2.23	0.63
<i>Trucks-TC-20</i>	(22,4)	4	—	209.12	1.00	12.77	—
<i>Pathways-PR-30</i>	(30,N/A)	30	100.91	37.32	14.37	6.98	1.00
<i>Pathways-MT-30</i>	(40,N/A)	40	587.23	213.34	54.31	12.89	1.00
<i>Rovers-PR-40</i>	(69,14)	14	—	6.98	5.01	2.78	3.44
<i>Rovers-MT-38</i>	(55,14)	14	—	—	—	2.89	—
<i>PipesWorld-PR-45</i>	(9,26)	9	—	11.73	0.89	1.34	1.00
<i>PipesWorld-MT-45</i>	(9,26)	9	—	11.12	0.91	1.28	1.00

b : # of bottleneck variables; g : # of guidance variables
 ‘—’: CPU time exceeding 30 minutes; N/A: No bottleneck variables

Table 4: Number of instances solved by SGPLan₅ with respect to the total number of instances (in parenthesis) in each domain variant. (— means no instances in that domain.)

Domain	Prop.	MetTime	Simp.	Qual.	Comp.	Const.
<i>TPP</i>	30(30)	79(80)	20(20)	20(20)	20(20)	18(30)
<i>OpenStacks</i>	30(30)	40(40)	20(20)	20(20)	—	—
<i>Trucks</i>	28(30)	30(30)	20(20)	20(20)	20(20)	20(20)
<i>Storage</i>	30(30)	30(30)	20(20)	20(20)	20(20)	9(30)
<i>Pathways</i>	30(30)	30(30)	30(30)	—	30(30)	—
<i>Rovers</i>	40(40)	32(40)	20(20)	20(20)	—	—
<i>PipesWorld</i>	30(50)	30(50)	—	—	15(18)	0(20)

addressed in the future is the design of a general strategy for handling such constraints with multiple objectives.

We have observed two interesting features of the *PipesWorld* domain in which SGPLan₅ does not perform well: the set of guidance variables overlaps with that of the bottleneck variables, and, unlike other domains, the bottleneck (*resp.*, guidance) variables have strong causal dependencies with each other. Further, the identified locality is strong only when the traffic density of the oil derivatives is low. We believe a better understanding of the constraint structure of this domain will lead a more refined partitioning strategy.

Finally, Figure 4 compares the performance of SGPLan₅ on six domain variants with that of other competing planners that participated in IPC5. (More complete results and information about these planners can be found at the competition Web site [Gerevini *et al.*, 2006].) All the experiments were conducted on a 3-GHz Intel Xeon Linux computer under a time limit of 30 minutes and a memory limit of 1 GBytes. The results show that SGPLan₅ is much faster than other competing planners and can solve many more instances. With respect to the plan metric value, SGPLan₅ is clearly the best planner. For PDDL2.2 domains, with the aid of new domain analysis techniques and a good partitioning strategy, SGPLan₅ can either solve more instances (*Storage-Propositional*) or achieve better plan quality (*OpenStacks-MetricTime*). The success of SGPLan₅ is attributed to the partitioning of large problems into more manageable subproblems that are related by a small number of global constraints.

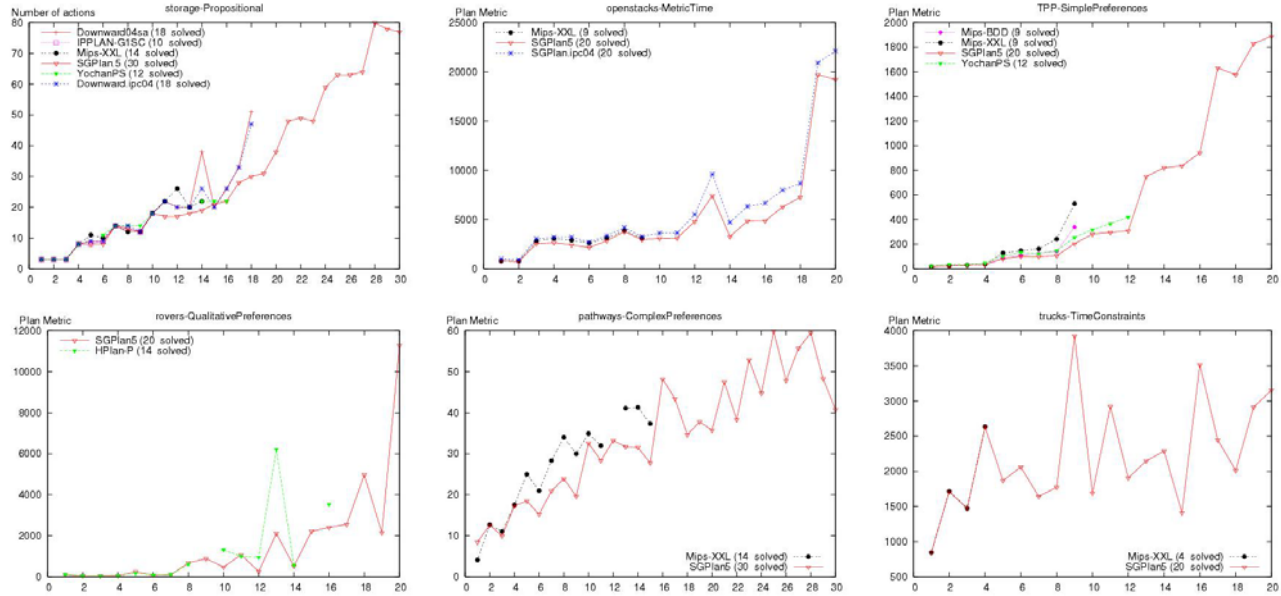


Figure 4: Performance comparison of SGPlan₅ and other competing planners on selected IPC5 domains (reproduced from results posted in [Gerevini *et al.*, 2006]).

5 Conclusions

In this paper we have proposed a new approach for partitioning the constraints of PDDL3 planning problems. Our improved approach exploits constraint locality better than that developed for solving PDDL2.2 problems. We also demonstrate the success of our planner on the IPC5 benchmarks.

Our future work will focus on developing techniques for supporting the new features in PDDL3. One possible direction is to enhance the efficiency of evaluating trajectory constraints. We will improve the heuristics in SGPlan₅ for handling complex trajectory constraints. We will also consider using the techniques in [Edelkamp, 2006] that translates trajectory constraints into some automata. With synchronized simulation of the automata, the evaluation of propositional trajectory constraints can be done by checking only the current state instead of all the intermediate states, although additional encoding space will be needed to keep track of all state transitions.

References

- [Bäckström and Nebel, 1995] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656, 1995.
- [Chen *et al.*, 2006] Y. X. Chen, B. W. Wah, and C. W. Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *J. of Artificial Intelligence Research*, (accepted to appear) Jan. 2006.
- [Edelkamp and Hoffmann, 2004] S. Edelkamp and J. Hoffmann. Classical part, 4th international planning competition. <http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/>, 2004.
- [Edelkamp, 2006] S. Edelkamp. On the compilation of plan constraints and preferences. In *ICAPS*, pages 374–377, 2006.
- [Gerevini and Long, 2005] A. Gerevini and D. Long. Plan constraints and preferences for PDDL3. Technical report, Technical report, R.T. 2005-08-07, Dept. of Electronics for Automation, U. of Brescia, Brescia, Italy, August 2005.
- [Gerevini *et al.*, 2006] A. Gerevini, Y. Dimopoulos, P. Haslum, and A. Saetti. Deterministic part, 5th international planning competition. <http://eracle.ing.unibs.it/ipc-5/>, 2006.
- [Helmert, 2004] M. Helmert. A planning heuristic based on causal graph analysis. In *ICAPS*, pages 161–170, 2004.
- [Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, 14:253–302, 2001.
- [Hoffmann *et al.*, 2004] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *J. of Artificial Intelligence Research*, 22:215–278, 2004.
- [Hoffmann, 2003] J. Hoffmann. The Metric-FF planning system: Translating ignoring delete lists to numeric state variables. *J. of Artificial Intelligence Research*, 20:291–341, 2003. <http://www.mpi-sb.mpg.de/~hoffmann/metric-ff.html>.
- [Hsu *et al.*, 2005] C. W. Hsu, B. W. Wah, and Y. X. Chen. Subgoal ordering and granularity control for incremental planning. In *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence*, pages 507–514, November 2005.
- [Wah and Chen, 2006] B. Wah and Y. X. Chen. Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*, 170(3):187–231, 2006.