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

Chih-Wei Hsu and Benjamin W. Wah
Dept. of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
{*chsu,wah*}*@manip.crhc.uiuc.edu*

Ruoyun Huang and Yixin Chen
Dept. of Computer Science and Engineering
Washington University in St Louis
St Louis, MO 63130, USA
rh11@cec.wustl.edu
chen@cse.wustl.edu

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 much 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 global constraints, optimizing goal preferences, and achieving subgoals in a multi-valued representation. Empirical results on the 5^{th} 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 makespan, are inadequate. Inspired by real applications, soft goals and trajectory constraints have been introduced in the PDDL3 specifications [Gerevini and Long, 2005].

Soft goals 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. Planning with soft goals entails the selection of an appropriate subset of the soft goals when it is infeasible to achieve all of them. In the worst case, an exhaustive search is needed 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 identify a good path among the many feasible paths to a goal state. Informally, for a plan trajectory $\pi = \langle (S_0,t_0),(S_1,t_1),\ldots,(S_n,t_n)\rangle$, 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. For a planning problem, we represent the actions scheduled as *variables* and identify *constraints* on mutual exclusion (mutex), goal state, and trajectory for actions in a given (possibly infeasible) plan. We then partition the constraints into subproblems, solve the subproblems individually by an existing but modified planner, and resolve those violated global constraints across the subproblems.

Constraint partitioning has been demonstrated to be a very effective technique for solving large planning problems [Chen *et al.*, 2006]. Because each subproblem only has a fraction of the original constraints and is a significant relaxation of the original problem, it is of much 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 [Wah and Chen, 2006].

Our constraint partitioning approach has four components: i) problem representation, ii) attributes used for partitioning, iii) partitioning algorithm, and iv) subproblem formulation. The design of each component depends on the problem representation and the planner used for solving the subproblems. Note that the variables and the constraints of a planning problem studied are not determined beforehand because the length of its solution plan is not known. Hence, we cannot directly partition its constraints by a clustering algorithm. Instead, we identify some problem-dependent attributes that can be used to characterize constraint localities in an initial (but possibly infeasible) plan. We then use these attributes to partition the constraints of the initial plan into subproblems with good localities before solving the subproblems.

For IPC4 benchmarks represented in PDDL2.2 [Edelkamp and Hoffmann, 2004], our original partition-and-resolve strategy in SGPlan$_4$ [Chen *et al.*, 2006] partitions the con-

[*]Research supported by the National Science Foundation Grant IIS 03-12084.

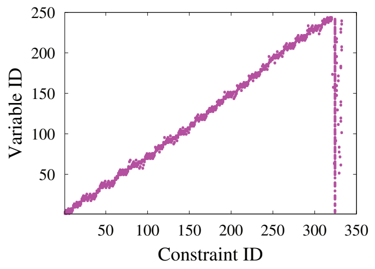


Figure 1: The constraint-variable structure depicting a strong constraint locality when the constraints of the IPC5 *Rovers-Propositional-30* instance is clustered by its subgoals.

straints of these benchmarks by their subgoals. The goals of many of these benchmarks are a conjunctive list of facts, with a strong locality of the mutex constraints by the subgoals of each problem. Here, *constraint locality* is measured by the fraction of constraints that are global, which span across multiple subproblems after the problem has been partitioned. Figure 1 shows a strong locality after clustering the constraints of the IPC5 *Rovers-Propositional-30* instance [Gerevini *et al.*, 2006] by its subgoals. The approach works well because the goal is indeed a conjunctive list of facts.

Subgoal partitioning, however, is not always effective for PDDL3 domains. The presence of soft goals and trajectory constraints may lead to constraint localities that are different from those 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 a poor constraint locality.

In clustering constraints of IPC5 instances, we observe that the propositional representation of binary facts in SGPlan$_4$ usually requires superfluous facts and obscures many mutexes. For example, the location of ***truck1*** in the *TPP* domain is represented by several binary facts, like ***at(truck1, market1)***, . . . , ***at(truck1, market8)***, with some implicit constraints for enforcing their consistency. To represent these constraints more compactly, 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 for the previous versions of PDDL, including Fast Downward [Helmert, 2004] and the IP planner [van den Briel *et al.*, 2005]. This representation makes implicit mutexes explicit and reduces the number of global constraints across the subproblems. For instance, the location of ***truck1*** can now be denoted by one variable with multiple values.

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 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. Our goal in this section is, therefore, to identify new problem-independent attributes that allow PDDL3 problems to be partitioned with better constraint locality.

Recall that a solution plan for a PDDL3 problem is represented in multi-valued variables and satisfies mutex, 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 into subproblems. Since the goal-state representation is also a conjunction of facts, our partition-and-resolve approach in SGPlan$_4$ can be applied directly.

Figure 2b illustrates the constraint-variable distribution when we cluster the mutex constraints (in 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 (in blue points) that are not localized.

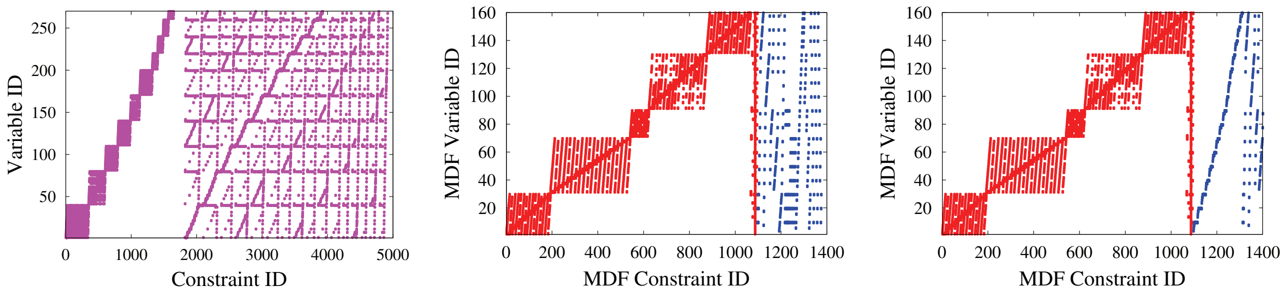
To address this issue, we further cluster the soft constraints by the same guidance variables. Figures 2c shows that all the constraints are now clustered well, leading to a good constraint locality. Note that we may have constraints that involve a limited number of guidance variables yet exhibit strong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

In addition to good locality, one advantage of using guidance variables is that they can be converted into classical planning goals. This greatly reduces the efforts of implementing a planner for solving a subproblem because current planners can only solve subproblems with nonempty and conflict-free goals. This also explains why we cannot arbitrarily construct subproblems with inconsistent goals or constraints using direct graph 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. The use of MDF does not introduce new types of constraints because any mutex in MDF must also be a mutex in PDDL.

We first define the guidance variables to be the set of variables in the goal-state constraints. Once the values of these



a) Naïve clustering by soft goals b) Clustering mutexes by guidance variables c) Clustering mutex and trajectory constraints

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 the constraints.

variables have been determined by satisfying all the hard goal constraints and by minimizing to some extent the penalties of the soft goal constraints, 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 limited objects are the sources of mutexes because actions want to concurrently access them. They can be treated as bottleneck resources that limit parallelism. Specifically, we search for a group of state variables of the same type (called *bottleneck variables*) that the changes of other state variables must depend on. For example, the locations of vehicles in the transportation domains are bottleneck variables because every action asserts the location of one vehicle. 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.

In addition to MDF analysis, we also identify and remove *accompanying state variables* in order to eliminate possibly redundant guidance and bottleneck variables. These are state variables whose values can be inferred from the values of other state variables. For example, ***on(crate1, depot0-1-1)*** implies ***in(crate1, depot0)***, given that ***in(depot0-1-1 depot0)*** is true (where ***depot0-1-1*** is a storage area in ***depot0***). This identification is possible because action *add* **on(crate1, depot0-1-1)** must also *add* **in(crate1, depot0)**, whereas *delete* **in(crate1, depot0)** must also *delete* **on(crate1, 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.

Table 1 shows the trade-offs in granularity on the IPC5 *Trucks-TimeConstraints-20* instance. When the instance is 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, whereas 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, whereas the number of global constraints is increased dramatically. In this case, partitioning is best done with respect to the bottleneck variables.

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

Finally, we cluster the guidance variables by formulating 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, ***(imply (stored goods11 level1) (stored goods18 level2))*** is true if we have determined the value of ***stored(goods11)*** to be ***level1***, and the value of ***stored(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* do-

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

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

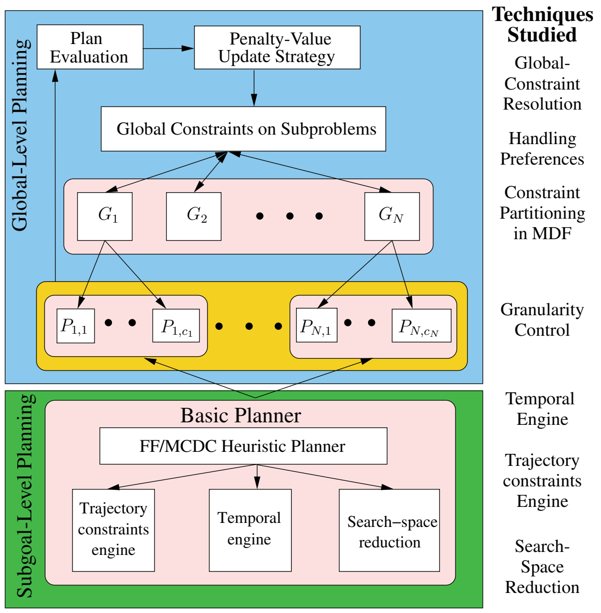


Figure 3: Architecture of SGPlan$_5$.

main 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.

3 Architecture of SGPlan$_5$

SGPlan$_5$ 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/bottlenecek 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

SGPlan$_5$ solves a planning problem formulated in one 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]. In each iteration, the planner solves a subproblem by generating a feasible plan that satisfies the local constraints as well as to minimize the weighted violations of the global constraints.

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:

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

where $\pi = \langle (S_0,t_0),\ldots,(S_n,t_n)\rangle$ defines the plan trajectory, $H(\pi)$ is the set of hard constraints, and $R(x)$ is the value of 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 those 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 (modified Metric-FF) 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 optimality because the basic planner does not ensure the optimality of $J_\pi(\pi)$.

Sometimes the search can be decomposed by the goal-state constraints, and the corresponding variables can be partitioned into disjoint sets. However, since the number of reachable assignments is exponential with respect to x, we generally limit the number of assignments explored when the space enumerated is still too large after the decomposition.

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 mutexes when handling global-level trajectory constraints. Since soft constraints do not have to be always satisfied, we compute the plan metric at the end of each iteration and record the incumbent. Note that the penalties used for resolving constraint violations differ from the preference weights used for computing the metric value.

3.2 Subproblem-Level Search

Our basic planner mainly follows Metric-FF's heuristic algorithm [Hoffmann, 2003], but employs a new heuristic function to better handle the new features in PDDL3. Using MDF, we have implemented a new heuristic that is 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 and without pruning the causal graphs. In contrast, the Fast Downward heuristic is incomplete since it prunes arcs from the causal graphs. Although MCDC provides a more accurate heuristic plan, its efficiency cannot compete with Metric-FF's "*ignore-delete-lists*" heuristic function. We only use MCDC as a better dead-end detector $R(x)$ and metric-value estimator $\tilde{J}(x, \pi)$.

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) by 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 the constraints or in optimizing the preferences.

To improve the efficiency of BFS, we incorporate new heuristics for constraint satisfaction into 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 the 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 resolving the global constraints. For instance, all the subproblems in the IPC5 *OpenStacks* domain are trivial to solve, but the 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 number 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 3: Slowdowns of solving some large IPC5 instances by some fixed number 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
TPP-PR-30	(20,8)	8	–	–	4.98	1.00	1.72
TPP-MT-40	(25,10)	10	–	16.35	2.92	2.01	1.29
TPP-MTC-22	(4,2)	2	–	1.00	–	–	–
OpenStacks-PR-30	(100,1)	1	1.00	3.34	4.09	3.34	5.01
OpenStacks-MT-30	(50,N/A)	50	0.76	0.76	0.76	0.76	1.00
Storage-PR-30	(20,5)	5	–	–	45.66	7.64	0.72
Storage-TI-30	(20,5)	5	–	–	40.68	6.15	9.23
Storage-TC-9	(9,5)	5	19.23	3.45	1.98	1.01	0.98
Trucks-PR-20	(20,1)	1	1.00	7.82	10.45	–	–
Trucks-TI-30	(32,5)	5	3.27	0.71	0.57	2.23	0.63
Trucks-TC-20	(22,4)	4	–	209.12	1.00	12.77	–
Pathways-PR-30	(30,N/A)	30	100.91	37.32	14.37	6.98	1.00
Pathways-MT-30	(40,N/A)	40	587.23	213.34	54.31	12.89	1.00
Rovers-PR-40	(69,14)	14	–	6.98	5.01	2.78	3.44
Rovers-MT-38	(55,14)	14	–	–	–	2.89	–
PipesWorld-PR-45	(9,26)	9	–	11.73	0.89	1.34	1.00
PipesWorld-MT-45	(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 SGPlan$_5$ with respect to the total number of instances (in parenthesis) in each variant. (– means no instances in that domain. *MetTime* for *TPP* and *OpenStacks* includes both *Metric* and *MetricTime*.)

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

Table 4 summarizes the statistics of the IPC5 instances solved by SGPlan$_5$ 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.

For the *MetricTimeConstraint* track, the weak performance of SGPlan$_5$ is due to the ineffective heuristics for handling trajectory constraints when one logical formula is before another, either quantitatively or qualitatively.

For the *PipesWorld* domain in which SGPlan$_5$ does not perform well, we have observed two interesting features. In this domain, 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. The identified locality is also weaker because the density of the shared resources is higher.

Finally, Figure 4 compares the performance of SGPlan$_5$ on six domain variants with that of other competing planners that participated in IPC5. 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 SGPlan$_5$ is much faster and can solve many more instances, when compared to other competing planners that do

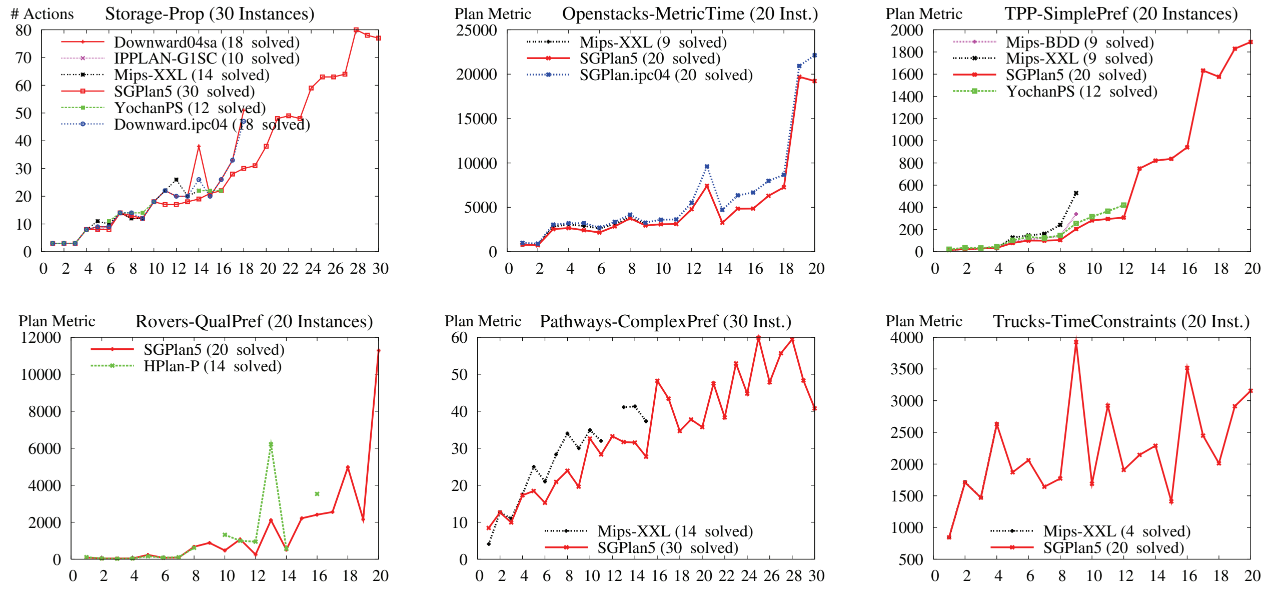


Figure 4: Performance comparison on selected domains of SGPlan$_5$ and other competing planners in the 5th International Planning Competitions (reproduced from the results posted in [Gerevini *et al.*, 2006]). (The x axis shows the instance number.)

not use constraint partitioning. With respect to the plan metric value, SGPlan$_5$ is clearly the best planner. For PDDL2.2 domains, with the aid of new domain analysis techniques and a good partitioning strategy, SGPlan$_5$ can either solve more instances (*Storage-Propositional*) or achieve better plan quality (*OpenStacks-MetricTime*).

The complete results at the IPC5 Web site [Gerevini *et al.*, 2006] show that SGPlan$_5$ ranks first (*resp.*, second) in 32 (*resp.*, 3) of the domain variants. The other competing planners that rank first/second in at least one of the domain variants include Downward (1/4), MIPS-BDD (0/1), MIPS-XXL (0/14), HPlan-B (0/5), and YochanPS (1/8).

In the future, we like to enhance the efficiency of evaluating trajectory constraints. We will improve the heuristics in SGPlan$_5$ for handling complex trajectory constraints. We will also consider using the techniques in [Edelkamp, 2006] that translates trajectory constraints into some automata in order to more efficiently check the satisfaction of constraints.

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*, 26:323–369, August 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, 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.

[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.

[van den Briel *et al.*, 2005] M. van den Briel, T. Vossen, and S. Kambhampati. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *ICAPS*, pages 310–319, 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.