Journal of Artificial Intelligence Research 26 (2006) 323-369 Submitted 10/05; published 08/06

Temporal Planning using Subgoal Partitioning and Resolution in SGPlan

Yixin Chen CHEN@CSE.WUSTL.EDU
Department of Computer Science and Engineering
Washington University in St Louis
St Louis, MO 63130 USA

Benjamin W. Wah WAH@MANIP.CRHC.UIUC.EDU
Chih-Wei Hsu CHSU@MANIP.CRHC.UIUC.EDU
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA

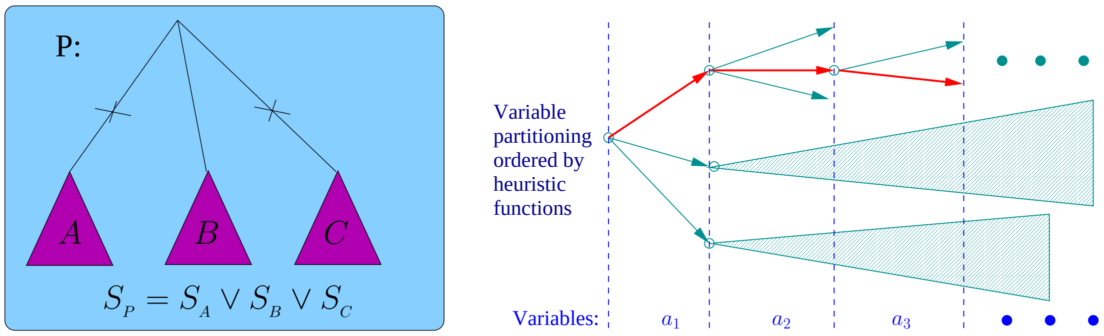
Abstract

In this paper, we present the partitioning of mutual-exclusion (mutex) constraints in temporal planning problems and its implementation in the SGPlan$_4$ planner. Based on the strong locality of mutex constraints observed in many benchmarks of the Fourth International Planning Competition (IPC4), we propose to partition the constraints of a planning problem into groups based on their subgoals. Constraint partitioning leads to significantly easier subproblems that are similar to the original problem and that can be efficiently solved by the same planner with some modifications to its objective function. We present a partition-and-resolve strategy that looks for locally optimal subplans in constraint-partitioned temporal planning subproblems and that resolves those inconsistent global constraints across the subproblems. We also discuss some implementation details of SGPlan$_4$, which include the resolution of violated global constraints, techniques for handling producible resources, landmark analysis, path finding and optimization, search-space reduction, and modifications of Metric-FF when used as a basic planner in SGPlan$_4$. Last, we show results on the sensitivity of each of these techniques in quality-time trade-offs and experimentally demonstrate that SGPlan$_4$ is effective for solving the IPC3 and IPC4 benchmarks.

1. Introduction

In this paper, we present an innovative partition-and-resolve strategy and its implementation in SGPlan$_4$ for solving temporal planning problems in PDDL2.2. Our strategy partitions the mutual-exclusion (mutex) constraints of a temporal planning problem by its subgoals into subproblems, solves the subproblems individually using a modified Metric-FF planner, and resolves those violated global constraints iteratively across the subproblems. We evaluate various heuristics for resolving global constraints and demonstrate the performance of SGPlan$_4$ in solving the benchmarks of the Third (IPC3) and the Fourth (IPC4) International Planning Competitions.

Most general and popular methods for solving large planning problems, such as systematic search, heuristic search, and transformation methods, can be viewed as the recursive

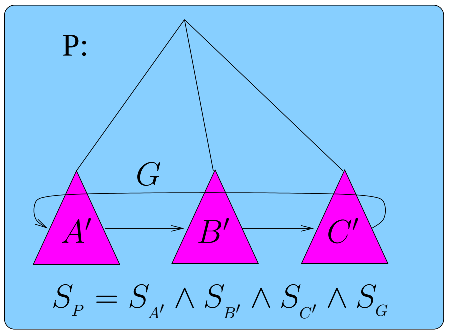

P:
A
B
C
$S_P = S_A \vee S_B \vee S_C$
Variable partitioning ordered by heuristic functions
Variables:
a_1
a_2
a_3

a) Search-space partitioning b) Complete and heuristic searches

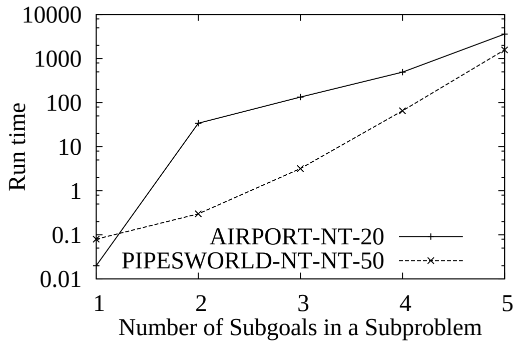
Figure 1: Search-space partitioning branches on variable assignments in order to decompose P into a disjunction ($\vee$) of subproblems with disjoint search spaces. The complexity of each subproblem is similar to that of P.

partitioning of a search space into independent subproblems and the iterative evaluation of the subproblems until a feasible solution is found. At each level of application of the approach, a problem or a subproblem is decomposed by partitioning its variable space into a disjunction ($\vee$) of subspaces (Figure 1a). To reduce the search complexity, the approach is often combined with intelligent backtracking that employs variable/value ordering to order the subproblems generated, that pre-filters partial inconsistent assignments to eliminate infeasible subproblems, and that prunes subproblems using bounds computed by relaxation or approximation.

Search-space partitioning can be directly applied on a planning problem or on a transformed version of the problem. Direct methods include complete and heuristic searches. As is illustrated in Figure 1b, these methods partition a search space recursively by branching on assigned variables (selection of actions). The difference between a complete search and a heuristic search is that the former enumerates all subspaces systematically, whereas the latter prioritizes subspaces by a heuristic function and evaluates them selectively. Examples of complete planners include UCPOP (Penberethy & Weld, 1992), Graphplan (Blum & Furst, 1997), STAN (Long & Fox, 1998), PropPLAN (Fourman, 2000), System R (Lin, 2001), SIPE-2 (Wilkins, 1990), O-Plan2 (Tate, Drabble, & Kirby, 1994), ZENO (Penberethy & Weld, 1994), TALplanner (Doherty & Kvarnstrm, 1999), and SHOP2 (Nau, Muoz-Avila, Cao, Lotem, & Mitchell, 2001); examples of heuristic planners include HSP (Bonet & Geffner, 2001), FF (Hoffmann & Nebel, 2001), AltAlt (Nigenda, Nguyen, & Kambhampati, 2000), GRT (Refanidis & Vlahavas, 2001), MO-GRT (Refanidis & Vlahavas, 2002), ASPEN (Chien, Rabideau, Knight, Sherwood, Engelhardt, Mutz, Estlin, Smith, Fisher, Barrett, Stebbins, & Tran, 2000), Metric-FF (Hoffmann & Nebel, 2001), GRT-R (Refanidis & Vlahavas, 2001), LPG (Gerevini & Serina, 2002), MIPS (Edelkamp, 2002), Sapa (Subbarao & Kambhampati, 2002), and Europa (Jonsson, Morris, Muscettola, & Rajan, 2000). In contrast, in a transformation approach, a problem is first transformed into a satisfiability or an optimization problem, before the transformed problem is solved by a SAT or integer programming solver that employs search-space partitioning. Notable planners using this



a) Constraint partitioning



b) Exponential behavior in solution time

Figure 2: Constraint partitioning decomposes P into a conjunction ($\wedge$) of subproblems with disjoint constraints but possibly overlapping search spaces, and a set of global constraints (G) to be resolved. Since the complexity of each subproblem is substantially smaller than that of P, it leads to an exponential decrease in solution time by Metric-FF on two IPC4 benchmarks (AIRPORT-NONTEMP-20 and PIPESWORLD-NOTANKAGE-NONTEMP-50) when the number of subgoals in a subproblem is decreased from 5 to 1.

approach include SATPLAN (Kautz & Selman, 1996), Blackbox (Kautz & Selman, 1999), ILP-PLAN (Kautz & Walser, 2000), and LPSAT (Wolfman & Weld, 2000).

One of the limitations of search-space partitioning is that the complexity of a problem is not dramatically reduced through partitioning. Although pruning and ordering strategies can make the search more efficient by not requiring the search of every subspace, the aggregate complexity of finding a solution to one of the subproblems is the same as that of the original problem.

In this paper, we study a *constraint-partitioning approach* that decomposes the constraints of a planning problem into a conjunction ($\wedge$) of subproblems with disjoint constraints but possibly overlapping search spaces (Figure 2a). The concept of constraints on planning problems studied in this paper is precisely defined in Section 2.1. Informally, a (mutex) constraint refers to the condition under which two actions can overlap with each other in their execution. Since all the constraints must be satisfied, all the subproblems must be solved in order to solve the original problem.

By decomposing the constraints of a problem into subproblems and by solving each independently, a subproblem will require significantly less time to solve because it is much more relaxed than the original problem. As an illustration, Figure 2b shows the exponential decrease in solution time when the number of subgoals in a subproblem is reduced linearly. Here, a *subgoal* is a collection of conjuncts in a conjunctive top-level goal of the problem. For both of the IPC4 instances evaluated, the run time is more than 1500 seconds when there are five subgoals in a subproblem and less than one second when there is one. Hence, the aggregate complexity of solving all the decomposed subproblems is exponentially smaller than that of the original problem.

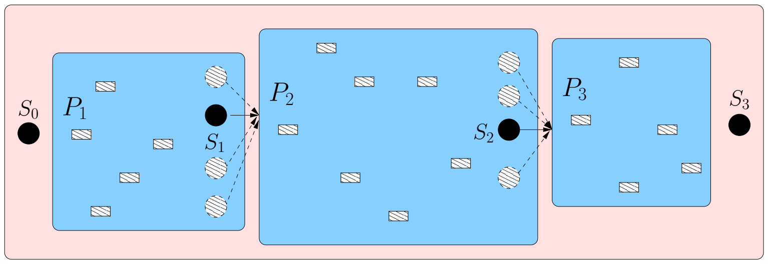


Figure 3: Partitioning the constraints of a planning problem along its temporal horizon into three stages requires finding suitable intermediate states S_1 and S_2 in order to connect the subplans in the three stages together. S_0 and S_3 are, respectively, the initial and the final states.

Constraint partitioning, however, leads to global constraints across subproblems (S_G in Figure 2a) that need to be resolved. These global constraints include those that span across common variables in multiple subproblems, such as those that relate two actions in different subproblems. Since these constraints may not be satisfied after solving the subproblems independently, the subproblems may need to be solved multiple times in order to resolve any violated global constraints.

In general, violated global constraints across subproblems cannot be efficiently resolved by brute-force enumeration because the search space for the global constraints is defined by the Cartesian product of the search spaces across the subproblems and is exponentially large. Dynamic programming cannot be applied because global constraints may span across multiple subproblems. This means that a partial feasible plan that dominates another partial feasible plan in one subproblem will fail to execute when the dominating plan violates a global constraint in another subproblem.

To address the resolution of violated global constraints, we summarize in Section 3 the theory of extended saddle points developed in our previous work (Wah & Chen, 2006). By choosing a suitable neighborhood, the theory allows a mixed-integer nonlinear programming problem (MINLP) to be partitioned into subproblems that are related by a necessary condition on the global constraints. Further, a necessary condition on each subproblem significantly prunes the Cartesian product of the search spaces across the subproblems in which inconsistent global constraints are to be resolved.

In addition to the efficient resolution of violated global constraints, the success of our approach depends on a strong locality of the constraints with respect to the actions they relate. We have observed informally in our previous work a strong locality of the constraints. Based on this strong locality, we have studied two alternatives for partitioning the constraints: partitioning them by time (Wah & Chen, 2006; Chen & Wah, 2003) and partitioning them by subgoals (Wah & Chen, 2004, 2003).

The idea of partitioning a planning problem by time is to partition its constraints by their temporal bindings into stages. To find an overall feasible plan, a planner will need to find a subplan from the initial to the final states of each stage that satisfy the local as well as the global constraints, where the final state of one stage will be the initial state of the next stage. For example, after partitioning the horizon into three stages (Figure 3), the planner assigns some values to the intermediate states S_1 and S_2, solves each subproblem individually, and perturbs S_1 and S_2 to look for another solution if feasible subplans cannot be found in any of the stages.

A major drawback of partitioning a planning problem by its temporal horizon is that constraint resolutions may have to sequentially propagate through multiple stages. We have found that the partitioning of the constraints in PDDL2.1 benchmarks along their temporal horizon often leads to many global constraints that only relate states in adjacent stages. As a result, when a violated subgoal is caused by an incorrect assignment of states in an early stage of the horizon, the resolution of the incorrect assignment will have to propagate sequentially through the stages. Oftentimes, the propagation of such information may cause a search to get stuck in an infeasible point for an extended period of time (Wah & Chen, 2004). To this end, an expensive enumeration of the final state in each stage (S_1 and S_2 in Figure 3) may be needed in order to resolve the inconsistencies.

A second approach we have studied in our previous work is to partition the constraints of a planning problem by their subgoals (Wah & Chen, 2004, 2003). After evaluating the subproblems, any inconsistent global constraints among them are first identified, and the subproblems are re-evaluated until all the global constraints are satisfied. Partitioning by subgoals eliminates the need of selecting a final state for each subproblem because the initial and the final states of each subgoal are known. Using this approach, our previous work has shown improvements in time and quality over the MIPS planner in solving some IPC3 benchmarks.

With respect to the second approach, we have made four main contributions in this paper.

First, we quantitatively evaluate in Section 2.2 the locality of constraints of all IPC4 benchmarks as well as benchmarks from the Blocksworld domain and the Depots domain. Our results show that constraint partitioning by subgoals consistently leads to a lower fraction of initial active global constraints than constraint partitioning by time. Our results also explain why constraint partitioning does not work well on some domains, such as Blocksworld and Depots.

Second, we incorporate Metric-FF (Hoffmann, 2003) as our basic planner in SGPlan$_4$ and SGPlan$_{4.1}$, instead of MIPS as in our previous work (Wah & Chen, 2004). This change is non-trivial because it requires significant extensions of Metric-FF in order to handle the new features in PDDL2.2 beyond those in PDDL2.1. These extensions include the support of temporal planning, the handling of derived predicates and timed initial literals, and the handling of wrappers for timed initial literals (Section 5.3).

Third, we describe new techniques for improving search efficiency in the global- and local-level architectures of our partition-and-resolve approach (Section 4.1). These include the handling of producible resources (Section 4.3), subgoal-level decomposition using landmark analysis, path finding and path optimization (Section 5.1), and subgoal-level planning using search-space reduction (Section 5.2). We explain their integration in our planners and analyze their effectiveness.

Last, we study in Section 4.2 trade-offs between solution time and quality in our heuristics for updating the penalties of violated global constraints. These trade-offs allow us to generate plans either of better quality but more time (SGPlan$_{4.1}$), or of lower quality but less time (SGPlan$_4$). The optimization of quality requires the estimation of the makespan of multiple subplans by an enhanced PERT algorithm (Section 5.3). In our previous work on constraint partitioning by subgoals (Wah & Chen, 2004), we have focused only on minimizing the planning time. Without optimizing quality, violated global constraints are often easier to resolve because a planner can always delay one or more actions in order to avoid such constraints. Finally, we compare in Section 7 the performance of our planners with respect to that of other planners.

2. Locality of Mutex Constraints in Temporal Planning

In this section, we define the mutex constraints of planning problems. Based on the structure of these constraints in IPC4 benchmarks, we show that constraint partitioning by subgoals leads to constraints that can be localized better than constraint partitioning by time.

2.1 Representation of Mutex Constraints

By following standard notations and definitions in the literature (Hoffmann & Nebel, 2001; Garrido, Fox, & Long, 2002), we summarize in this section the basic definitions of mutex constraints used in this paper.

Definition 1. A *planning problem* $\mathcal{T} = (\mathcal{O}, \mathcal{F}, \mathcal{I}, \mathcal{G})$ is a quadruple, where $\mathcal{O}$ is the set of possible actions in $\mathcal{T}$, $\mathcal{F}$ is the set of all facts, $\mathcal{I}$ is the set of initial facts, and $\mathcal{G}$ is the set of goal facts.

Definition 2. A *state* $S = \left\{f_1, \cdots, f_{n_S}\right\}$ is a subset of facts in $\mathcal{F}$ that are true.

Definition 3. A *STRIPS action* $a \in \mathcal{O}$ is associated with the following attributes:
a) $pre(a)$, a set of facts that define the preconditions of action a;
b) $add(a)$, a set of facts that define the add effects of a; and
c) $del(a)$, a set of facts that define the delete effects of a.
The resulting state of applying action a to state S is defined as:

$$Result(S, a) = \begin{cases} (S \bigcup add(a)) \backslash del(a) & \text{if } pre(a) \subseteq S \\ S & \text{if } pre(a) \nsubseteq S. \end{cases} \tag{1}$$

The resulting state of applying a sequence of actions $a_1, \cdots, a_n$ to S is recursively defined as:

$$Result(S, (a_1, \cdots, a_n)) = Result(Result(S, (a_1, \cdots, a_{n-1})), a_n). \tag{2}$$

Next, we extend our action model to temporal planning. For durative actions supported in PDDL2.2, a precondition fact can be effective at the beginning, at the end, or during the entire duration of an action; whereas an add effect or a delete effect can be effective only at the beginning or at the end of an action.

Definition 4. A *temporal action* $a \in \mathcal{O}$ is associated with the following attributes:
a) $s(a)$ and $e(a)$ define, respectively, the start time and the end time of a.

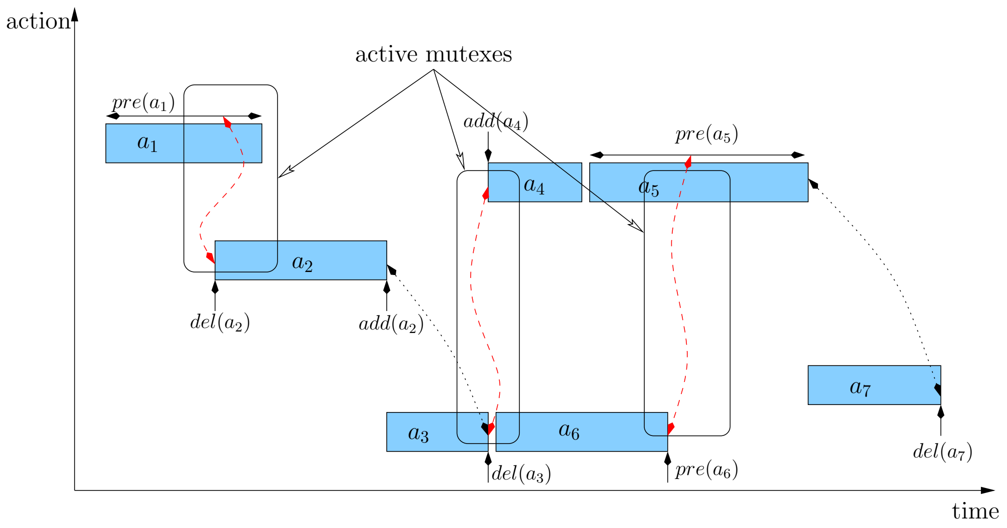


Figure 4: An example temporal plan, where active mutexes between actions are shown as dashed lines, and inactive mutexes as dotted lines.

b) The preconditions can be divided into three types: $pre_{start}(a)$, the set of initial preconditions to be held at $s(a)$; $pre_{end}(a)$, the set of final preconditions to be held at $e(a)$; and $pre_{overall}(a)$, the set of invariant preconditions over an open interval $(s(a), e(a))$.

c) There are two types of add effects: $add_{start}(a)$, the set of initial add effects to be asserted at $s(a)$; and $add_{end}(a)$, the set of final add effects to be asserted at $e(a)$.

d) There are two type of delete effects: $del_{start}(a)$, the set of initial delete effects to be asserted at $s(a)$; and $del_{end}(a)$, the set of final delete effects to be asserted at $e(a)$.

Definition 5. A *temporal plan* $\mathcal{P} = \{a_1, a_2, \cdots, a_m\}$ is a list of m temporal actions, where a_i has been assigned start time $s(a_i)$ and end time $e(a_i)$.

Figure 4 illustrates a temporal plan of seven actions. In each action, we indicate, where appropriate, its preconditions, add effects, and delete effects.

Concurrent actions in a plan must be arranged in such a way that observes mutual exclusions (mutexes). The notion of mutex was first proposed in GraphPlan (Blum & Furst, 1997). It was defined for a planning graph, which is a level-by-level constraint graph that alternates between a fact level and an action level. Mutex relationships in a planning graph can be classified into transient (level-dependent) and persistent (level-independent) (Blum & Furst, 1997). A mutex is *transient* if it exists only in certain levels of the graph and vanishes as more levels of the graph are built. In contrast, a mutex is *persistent* if it holds at every level until the fix-point level (the last level of the graph) is achieved. In this paper, we only consider level-independent, persistent mutex relationships, as transient mutexes are exclusively used for searches in GraphPlan.

Actions a and b are marked as *persistently mutual exclusive* when one of the following occurs.

a) Actions a and b have persistent *competing needs*,[1] in which competing needs are represented by the persistent mutex of the preconditions of a and those of b;

b) They have persistent *inconsistent effects*, when one action deletes an add effect of the other.

c) They have persistent *interference*, when one action deletes a precondition of the other.

Two facts p and q are *persistently mutual exclusive* if all possible ways of making p true are persistently exclusive with all possible ways of making q true; that is, each action a having p as an add effect ($p \in add(a)$) is persistently mutual exclusive with each action b having q as an add effect ($q \in add(b)$). For simplicity, in the rest of this paper, mutex actions and facts refer to the corresponding persistent mutex actions and facts.

Given a temporal plan, a mutex relationship can be *active* or *inactive*. For example, actions a_1 and a_2 in Figure 4 have an active mutex because the two actions overlap in their execution and have persistent interference. However, a_2 and a_3 have an inactive mutex because they do not overlap in their execution.

Based on the above discussion, the conditions for an active mutex to occur between two actions a and b can be summarized in four cases (Garrido et al., 2002):

a) Actions a and b start together, and there is a nonempty intersection between their initial preconditions (*resp.* add effects) and their initial delete effects (*resp.* delete effects).

b) Actions a and b end together, and there is a nonempty intersection between their final preconditions (*resp.* add effects) and their final delete effects (*resp.* delete effects).

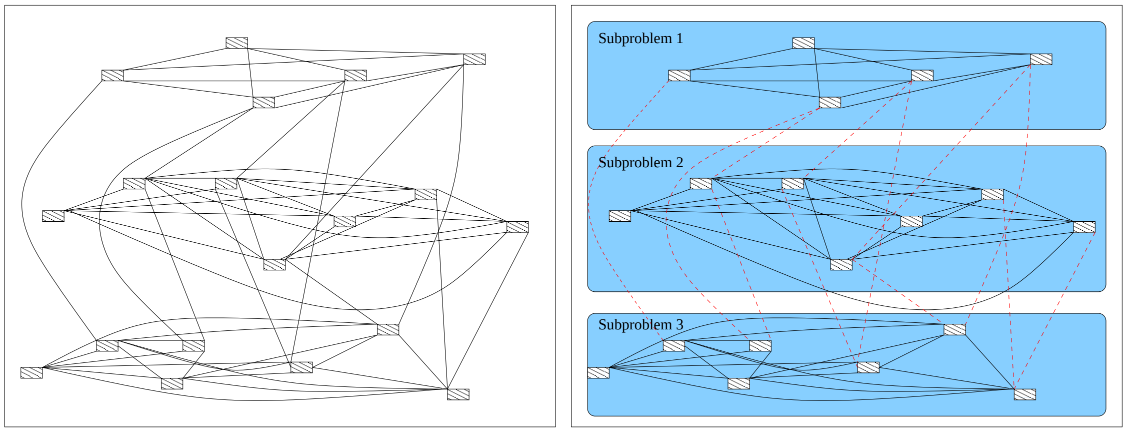
c) Action a ends when b starts, and there is a nonempty intersection between the final delete effects (*resp.* delete effects, add effects, and preconditions) of a and the initial add effects (*resp.* preconditions, delete effects, and delete effects) of b.

d) Action a starts (*resp.* ends) during the execution of b, and there is a nonempty intersection between the initial (*resp.* final) delete effects of a and the invariant preconditions of b.

While the conditions above are introduced to prevent two mutually exclusive actions from executing simultaneously, there may be actions that block the propagation of facts (no-op action) and that cause unsupported actions later. Such a condition can be detected by looking for actions that delete some existing facts in the current plan. With respect to conditions for mutex due to competing needs, we do not need to represent them explicitly because mutexes due to competing needs must accompany the other two types of mutex: when two preconditions are mutually exclusive due to competing needs, the two action sequences of making them true are also mutually exclusive. As example, the active mutex between a_5 and a_6 in Figure 4 is due to competing needs and is caused by the active mutex between a_3 and a_4.

The mutex constraints studied in this paper are not in closed form. Instead, each is defined by a discrete procedural function that checks if a pair of actions meet one of the four conditions above. The inputs to the function are the start time and the end time of each action, which are continuous in temporal problems and discrete in propositional problems.

1. The terms "competing needs," "inconsistent effects," and "interference" were originally proposed for GraphPlan (Blum & Furst, 1997).



a) 63 mutex constraints among actions   b) Partitioning the mutex constraints by subgoals

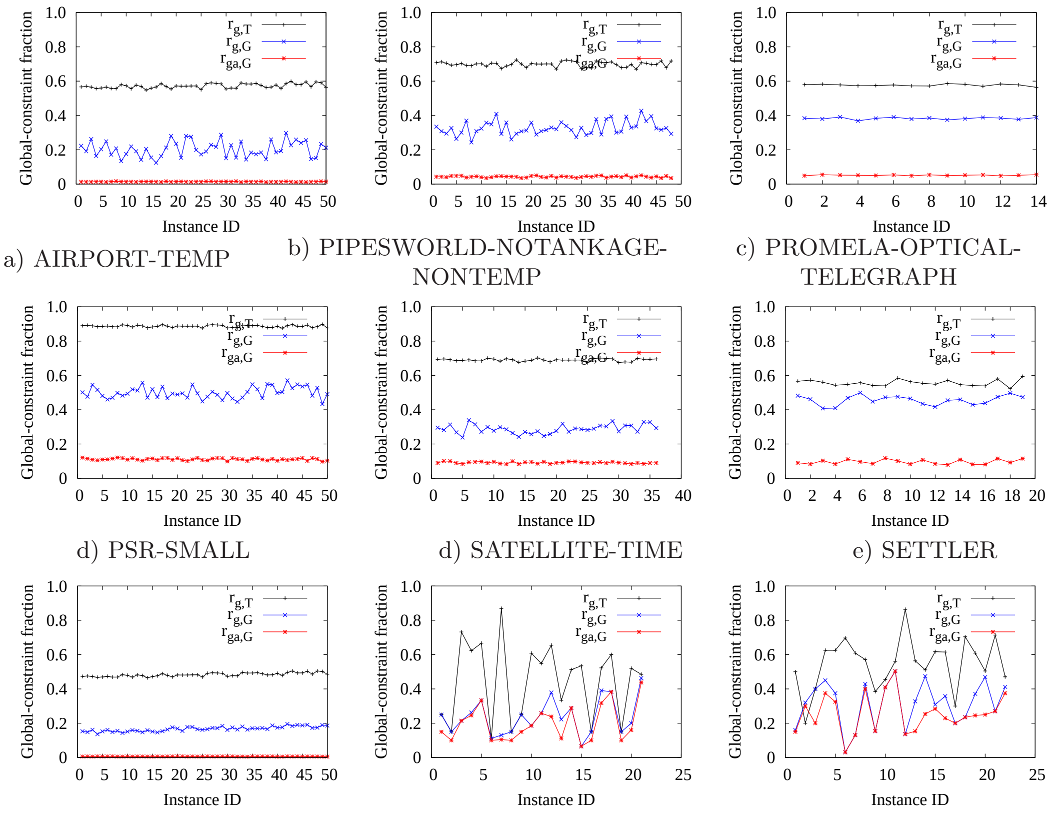
Figure 5: Mutex constraints in the IPC4 AIRPORT-TEMP-4 instance. Each rectangular box represents an action, and a line joining two actions represents a mutex constraint (that may be inactive). Most constraints (52 out of 63 or 83%) are local constraints after partitioning them by subgoals. Global mutex constraints are shown in dashed lines in (b).

2.2 Locality of Mutex Constraints

In this section, we evaluate the partitioning of mutex constraints for some planning benchmarks. Our analysis shows the strong locality of these constraints when they are partitioned by subgoals as compared to the case when they are partitioned by time. We do not study other criteria for partitioning because they may lead to subproblems whose initial and final states are not specified. Such subproblems will be hard to solve by existing planners because they may require a systematic enumeration of their initial and final states when finding feasible plans.

Figure 5a shows the 63 mutex constraints in a solution plan to the fourth instance of the IPC4 AIRPORT-TEMP domain. The instance involves moving three planes in an airport to designated gates. Each rectangular box in the figure represents an action, whereas a line joining two actions represents a mutex constraint (that may be inactive). Figure 5b shows the partitioning of the constraints into three subproblems, each involving the movement of one plane. We show local constraints (those that are relevant to the actions in one subproblem) in solid lines and global constraints relating those actions in different subproblems in dashed lines. It is clear that a majority (83%) of the constraints are local after partitioning them by subgoals.

To demonstrate the localization of mutex constraints when partitioned by subgoals, we analyze all the IPC4 instances. We first modify the original Metric-FF planner (Hoffmann, 2003) in order to support the new features in PDDL2.2, such as temporal actions and derived predicates. For each instance, we use the modified planner to find an initial subplan for each of the subproblems. We then find all the mutexes among the actions, including active and inactive ones. Finally, we compute the number of global constraints related to actions



a) AIRPORT-TEMP b) PIPESWORLD-NOTANKAGE-NONTEMP c) PROMELA-OPTICAL-TELEGRAPH

d) PSR-SMALL d) SATELLITE-TIME e) SETTLER

f) UMTS-TEMP g) DEPOTS-TIME h) BLOCKSWORLD

Figure 6: Variations of $r_{g,T}$, $r_{g,G}$, and $r_{ga,G}$ across the instances of seven IPC4 domain variants as well as the instances of the DEPOTS-TIME domain variant from IPC3 and those of the Blocksworld domain from IPC2. (The latter two domains are deemed difficult for constraint partitioning.)

in different subplans, as well as the number of initial active global constraints based on the subplan evaluated for each subproblem. As a comparison, we also evaluate the partitioning of the constraints by their temporal horizon.

Figure 6 illustrates the results for seven IPC4 domain variants, as well as the Blocksworld domain from IPC2 and the DEPOTS-TIME variant from IPC3. Table 1 further summarizes the average statistics across all the instances in each IPC4 domain variant and those of the Blocksworld domain and the Depots domain variants. For each instance in partitioning by time, we use the modified Metric-FF planner to find an initial plan, set the number of temporal stages to be the same as the number of subgoals, and partition the horizon of the solution plan evenly into multiple stages. We then count the number of local constraints in each stage and the number of global constraints relating actions in different stages. For each instance, let N_c be the total number of mutex constraints, N_g^T be the number of global constraints under constraint partitioning by time, N_g^G be the number of global constraints

Table 1: Average $r_{g,T}$, $r_{g,G}$, $r_{ga,G}$ across the instances of IPC4 domains as well as the Depots domain from IPC3 and the Blocksworld domain from IPC2. (The latter two are deemed difficult for constraint partitioning.) Boxed numbers are less than 0.1.

Domain Variant	$\overline{r}_{g,T}$	$\overline{r}_{g,G}$	$\overline{r}_{ga,G}$
AIRPORT-NONTEMP	0.557	0.219	$\boxed{0.017}$
AIRPORT-TEMP	0.568	0.208	$\boxed{0.014}$
AIRPORT-TEMP-TIMEWINDOWS	0.494	0.184	$\boxed{0.013}$
AIRPORT-TEMP-TIMEWINDOWS-CO	0.495	0.188	$\boxed{0.014}$
PIPESWORLD-NOTANKAGE-NONTEMP	0.695	0.313	$\boxed{0.044}$
PIPESWORLD-NOTANKAGE-TEMP	0.682	0.301	$\boxed{0.042}$
PIPESWORLD-NOTANKAGE-TEMP-DEADLINE	0.674	0.297	$\boxed{0.033}$
PIPESWORLD-TANKAGE-NONTEMP	0.687	0.677	$\boxed{0.070}$
PIPESWORLD-TANKAGE-TEMP	0.683	0.459	0.126
PIPESWORLD-NOTANKAGE-TEMP-DEADLINE-CO	0.682	0.296	$\boxed{0.039}$
PROMELA-OPTICAL-TELEGRAPH	0.575	0.399	$\boxed{0.052}$
PROMELA-OPTICAL-TELEGRAPH-DP	0.759	0.265	$\boxed{0.020}$
PROMELA-OPTICAL-TELEGRAPH-FL	0.799	0.426	$\boxed{0.037}$
PROMELA-PHILOSOPHER	0.554	0.370	$\boxed{0.066}$
PROMELA-PHILOSOPHER-DP	0.855	0.576	$\boxed{0.019}$
PROMELA-PHILOSOPHER-FL	0.822	0.507	$\boxed{0.087}$
PSR-SMALL	0.897	0.489	0.114
PSR-MIDDLE	0.896	0.504	$\boxed{0.092}$
PSR-MIDDLE-CO	0.882	0.478	$\boxed{0.049}$
PSR-LARGE	0.902	0.665	$\boxed{0.096}$
SATELLITE-STRIPS	0.689	0.288	$\boxed{0.096}$
SATELLITE-TIME	0.686	0.289	$\boxed{0.093}$
SATELLITE-TIME-TIMEWINDOWS	0.648	0.114	$\boxed{0.027}$
SATELLITE-TIME-TIMEWINDOWS-CO	0.633	0.307	$\boxed{0.075}$
SATELLITE-NUMERIC	0.288	0.305	$\boxed{0.078}$
SATELLITE-COMPLEX	0.642	0.282	$\boxed{0.069}$
SATELLITE-COMPLEX-TIMEWINDOWS	0.633	0.124	$\boxed{0.041}$
SATELLITE-COMPLEX-TIMEWINDOWS-CO	0.698	0.153	$\boxed{0.042}$
SETTLERS	0.549	0.451	0.100
UMTS-TEMP	0.463	0.157	$\boxed{0.006}$
UMTS-TEMP-TIMEWINDOWS	0.437	0.126	$\boxed{0.008}$
UMTS-TEMP-TIMEWINDOWS-CO	0.407	$\boxed{0.098}$	$\boxed{0.008}$
UMTS-FLAW-TEMP	0.459	0.136	$\boxed{0.006}$
UMTS-FLAW-TEMP-TIMEWINDOWS	0.428	0.110	$\boxed{0.008}$
UMTS-FLAW-TEMP-TIMEWINDOWS-CO	0.414	0.086	$\boxed{0.007}$
DEPOTS-STRIPS	0.537	0.418	0.231
DEPOTS-SIMPLETIME	0.572	0.304	0.167
DEPOTS-NUMERIC	0.491	0.354	0.188
DEPOTS-TIME	0.448	0.237	0.197
BLOCKSWORLD	0.549	0.314	0.254

under constraint partitioning by subgoals, and N_{ga}^G be the number of initial active global constraints under constraint partitioning by subgoals. We then compute the following ratios:

$$r_{g,T} = \frac{N_g^T}{N_c} : \text{ fraction of global constraints under constraint partitioning by time;}$$
$$r_{g,G} = \frac{N_g^G}{N_c} : \text{ fraction of global constraints under constraint partitioning by subgoals;}$$
$$r_{ga,G} = \frac{N_{ga}^G}{N_c} : \text{ fraction of initial active global constraints under subgoal partitioning.}$$

With respect to instances in the IPC4 domains, the results show that constraint partitioning by subgoals leads to a lower $r_{g,G}$ than $r_{g,T}$, that the fractions do not vary significantly, and that $r_{ga,G}$ is small for most instances. Except for PSR-SMALL and SETTLERS, $r_{ga,G}$ is consistently less than 0.1. This behavior is important because only active constraints will need to be resolved during planning, and the number of such constraints should decrease as planning progresses. We describe in Section 4.2 two strategies for reducing the number active global constraints in planning.

The behavior is worse for the instances in the Blocksworld domain and the Depots domain variants. In these two domains, $r_{ga,G}$ is consistently high (over 20%) when constraints are partitioned by subgoals. The reason is that the actions in different subgoals of each instance are highly related, making it more difficult to cluster the constraints and leading to a larger fraction of global constraints. We evaluate the performance of our approach on these two domains in Section 7.

3. Constraint Partitioning using Penalty Formulations

Given a constrained formulation of a planning problem, we summarize in this section our theory of extended saddle points in mixed space (Wah & Chen, 2006) that the design of our planners is based upon.

3.1 The Extended Saddle-Point Condition

Consider the following MINLP with variable $z = (x, y)$, $x \in \mathbb{R}^v$ and $y \in \mathbb{D}^w$:

$$(P_m): \qquad \min_z \quad f(z), \qquad \text{subject to} \quad h(z) = 0 \ \text{ and } \ g(z) \le 0, \tag{3}$$

where f is continuous and differentiable with respect to x, and $g = (g_1, \ldots, g_r)^T$ and $h = (h_1, \ldots, h_m)^T$ are general functions that are not necessarily continuous and differentiable. These assumptions are important because the constraints in our planners are procedural functions not in closed form. We further assume that f is lower bounded, while g and h can be unbounded.

The goal of solving P_m is to find a constrained local minimum $z^* = (x^*, y^*)$ with respect to $\mathcal{N}_m(z^*)$, the mixed neighborhood of z^*. Because the results have been published earlier (Wah & Chen, 2006), we only summarize some high-level concepts without the precise formalism.

Definition 6. A mixed neighborhood $\mathcal{N}_m(z)$, $z = (x, y)$, in mixed space $\mathbb{R}^v \times \mathbb{D}^w$ is:

$$\mathcal{N}_m(z) = \Big\{(x', y) \mid x' \in \mathcal{N}_c(x)\Big\} \cup \Big\{(x, y') \mid y' \in \mathcal{N}_d(y)\Big\}, \tag{4}$$

where $\mathcal{N}_c(x) = \{x' : \|x' - x\| \le \epsilon \text{ and } \epsilon \to 0\}$ is the continuous neighborhood of x, and the discrete neighborhood $\mathcal{N}_d(y)$ is a *finite* user-defined set of points $\{y' \in \mathbb{D}^w\}$.

Definition 7. Point z^* is a CLM_m, a constrained local minimum of P_m with respect to points in $\mathcal{N}_m(z^*)$, if z^* is feasible and $f(z^*) \le f(z)$ for all feasible $z \in \mathcal{N}_m(z^*)$.

Definition 8. The penalty function of P_m with penalty vectors $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^r$ is:

$$L_m(z, \alpha, \beta) = f(z) + \alpha^T |h(z)| + \beta^T \max(0, g(z)). \tag{5}$$

Next, we define informally a constraint-qualification condition needed in the main theorem (Wah & Chen, 2006). Consider a feasible point $z' = (x', y')$ and a neighboring point $z'' = (x' + \vec{p}, y')$ under an infinitely small perturbation along direction $\vec{p} \in X$ in the x subspace. When the constraint-qualification condition is satisfied at z', it means that there is no $\vec{p}$ such that the rates of change of all equality and active inequality constraints between z'' and z' are zero. To see why this is necessary, assume that $f(z)$ at z' decreases along $\vec{p}$ and that all equality and active inequality constraints at z' have zero rates of change between z'' and z'. In this case, it is not possible to find some finite penalty values for the constraints at z'' in such a way that leads to a local minimum of the penalty function at z' with respect to z''. Hence, if the above scenario were true for some $\vec{p}$ at z', then it is not possible to have a local minimum of the penalty function at z'. In short, constraint qualification at z' requires at least one equality or active inequality constraint to have a non-zero rate of change along each direction $\vec{p}$ at z' in the x subspace.

Theorem 1. Necessary and sufficient ESPC on CLM_m of P_m (Wah & Chen, 2006). Assuming $z^* \in \mathbb{R}^v \times \mathbb{D}^w$ of P_m satisfies the constraint-qualification condition, then z^* is a CLM_m of P_m iff there exist finite $\alpha^* \ge 0$ and $\beta^* \ge 0$ that satisfies the following *extended saddle-point condition* (ESPC):

$$L_m(z^*, \alpha, \beta) \;\le\; L_m(z^*, \alpha^{**}, \beta^{**}) \;\le\; L_m(z, \alpha^{**}, \beta^{**}) \tag{6}$$

for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$ and for all $z \in \mathcal{N}_m(z^*)$, $\alpha \in \mathbb{R}^m$, and $\beta \in \mathbb{R}^r$.

Note that (6) can be satisfied under rather loose conditions because it only requires any α^{**} and β^{**} that are larger than some critical α^* and β^*. The theorem is important because it establishes a one-to-one correspondence between a CLM_m z^* of P_m and an ESP (*extended saddle point*) of the corresponding unconstrained penalty function in (5) when the penalties are sufficiently large. The theorem also leads to an easy way for finding CLM_m. Since an ESP is a local minimum of (5) (but not the converse), z^* can be found by gradually increasing the penalties of those violated constraints in (5) and by repeatedly finding the local minima of (5) until a feasible solution to P_m is obtained. This is possible because there exist many algorithms for locating the local minima of unconstrained functions.

3.2 The Partitioned Extended Saddle-Point Condition

An important feature of the ESPC in Theorem 1 is that the condition can be partitioned in such a way that each subproblem implementing a partitioned condition can be solved by looking for any α^{**} and β^{**} that are larger than α^* and β^*.

Consider P_t, a version of P_m whose constraints can be partitioned into N subproblems:

$$\begin{array}{llllll} (P_t): & \min\limits_z & J(z) & & & \\ & \text{subject to} & h^{(t)}(z(t)) = 0, & g^{(t)}(z(t)) \le 0 & \text{(local constraints)} & (7) \\ & \text{and} & H(z) = 0, & G(z) \le 0 & \text{(global constraints).} & \end{array}$$

Subproblem t, $t = 1, \ldots, N$, of P_t has local *state vector* $z(t) = (z_1(t), \ldots, z_{u_t}(t))^T$ of u_t mixed variables, where $\cup_{t=1}^{N} z(t) = z$. Here, $z(t)$ includes all variables that appear in any of the m_t local equality constraint functions $h^{(t)} = (h_1^{(t)}, \ldots, h_{m_t}^{(t)})^T$ and the r_t local inequality constraint functions $g^{(t)} = (g_1^{(t)}, \ldots, g_{r_t}^{(t)})^T$. Since the partitioning is by constraints, $z(1), \ldots, z(N)$ may overlap with each other. $H = (H_1, \ldots, H_p)^T$ and $G = (G_1, \ldots, G_q)^T$ are global-constraint functions of z. We assume that J is continuous and differentiable with respect to its continuous variables, that f is lower bounded, and that g, h, G, and H are general functions that can be discontinuous, non-differentiable, and unbounded.

We first define $\mathcal{N}_p(z)$, the mixed neighborhood of z for P_t, and decompose the ESPC in (6) into a set of necessary conditions that collectively are sufficient. Each partitioned condition is then satisfied by finding the local ESP of a subproblem, and any violated global constraints are resolved by using appropriate penalties.

Definition 9. $\mathcal{N}_p(z)$, the *mixed neighborhood* of z for a partitioned problem, is:

$$\mathcal{N}_p(z) = \bigcup_{t=1}^{N} \mathcal{N}_p^{(t)}(z) = \bigcup_{t=1}^{N} \left\{ z' \;\middle|\; z'(t) \in \mathcal{N}_m(z(t)) \text{ and } z'_i = z_i \;\forall z_i \notin z(t) \right\}, \tag{8}$$

where $\mathcal{N}_m(z(t))$ is the mixed neighborhood of $z(t)$.

Intuitively, $\mathcal{N}_p(z)$ is separated into N neighborhoods, where the tth neighborhood perturbs only the variables in $z(t)$ while leaving those variables in $z \backslash z(t)$ unchanged.

Without showing the details, we can consider P_t as a MINLP and apply Theorem 1 to derive the ESPC of P_t. We then decompose the ESPC into N necessary conditions, one for each subproblem, and an overall necessary condition on the global constraints across the subproblems. We first define the penalty function for Subproblem t.

Definition 10. Let $\Phi(z, \gamma, \eta) = \gamma^T |H(z)| + \eta^T \max(0, G(z))$ be the sum of the transformed global constraint functions weighted by their penalties, where $\gamma = (\gamma_1, \ldots, \gamma_p)^T \in \mathbb{R}^p$ and $\eta = (\eta_1, \ldots, \eta_q)^T \in \mathbb{R}^q$ are the penalty vectors for the global constraints. Then the penalty function for P_t in (7) and the corresponding penalty function in Subproblem t are defined as follows:

$$L_m(z, \alpha, \beta, \gamma, \eta) = J(z) + \sum_{t=1}^{N} \left\{ \alpha(t)^T |h^{(t)}(z(t))| + \beta(t)^T \max(0, g^{(t)}(z(t)) \right\} + \Phi(z, \gamma, \eta), \tag{9}$$

$$\Gamma_m(z, \alpha(t), \beta(t), \gamma, \eta) = J(z) + \alpha(t)^T |h^{(t)}(z(t))| + \beta(t)^T \max(0, g^{(t)}(z(t))) + \Phi(z, \gamma, \eta), \tag{10}$$

where $\alpha(t) = (\alpha_1(t), \ldots, \alpha_{m_t}(t))^T \in \mathbb{R}^{m_t}$ and $\beta(t) = (\beta_1(t), \ldots, \beta_{r_t}(t))^T \in \mathbb{R}^{r_t}$ are the penalty vectors for the local constraints in Subproblem t.

Theorem 2. *Partitioned necessary and sufficient ESPC on CLM_m of P_t* (Wah & Chen, 2006). Given $\mathcal{N}_p(z)$, the ESPC in (6) can be rewritten into $N+1$ necessary conditions that, collectively, are sufficient:

$$\Gamma_m(z^*, \alpha(t), \beta(t), \gamma^{**}, \eta^{**}) \;\le\; \Gamma_m(z^*, \alpha(t)^{**}, \beta(t)^{**}, \gamma^{**}, \eta^{**}) \;\le\; \Gamma_m(z, \alpha(t)^{**}, \beta(t)^{**}, \gamma^{**}, \eta^{**}), \quad (11)$$

$$L_m(z^*, \alpha^{**}, \beta^{**}, \gamma, \eta) \;\le\; L_m(z^*, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}), \quad (12)$$

for any $\alpha(t)^{**} > \alpha(t)^* \ge 0$, $\beta(t)^{**} > \beta(t)^* \ge 0$, $\gamma^{**} \ge \gamma^* \ge 0$, and $\eta^{**} \ge \eta^* \ge 0$, and for all $z \in \mathcal{N}_p^{(t)}(z^*)$, $\alpha(t) \in \mathbb{R}^{m_t}$, $\beta(t) \in \mathbb{R}^{r_t}$, $\gamma \in \mathbb{R}^p$, $\eta \in \mathbb{R}^q$, and $t = 1, \ldots, N$.

Theorem 2 shows that the original ESPC in Theorem 1 can be partitioned into N necessary conditions in (11) and an overall necessary condition in (12) on the global constraints across the subproblems. The partitioned condition in Subproblem t can be satisfied by finding the ESPs in that subproblem. Because finding an ESP is equivalent to solving a MINLP, we can reformulate the search in Subproblem t as the solution of the following optimization problem:

$$\left(P_t^{(t)}\right): \qquad \min_{z(t)} \quad J(z) + \gamma^T |H(z)| + \eta^T \max(0, G(z)) \quad (13)$$
$$\text{subject to} \quad h^{(t)}(z(t)) = 0 \quad \text{and} \quad g^{(t)}(z(t)) \le 0.$$

The weighted sum of global constraint functions in the objective of $P_t^{(t)}$ is important because it leads to points that minimize the violations of global constraints. When γ^T and η^T are large enough, solving $P_t^{(t)}$ will lead to points, if they exist, that satisfy the global constraints.

In short, finding solutions of P_t that satisfy (6) can be reduced to solving multiple subproblems, where (13) can be solved by an existing solver with some modifications of the objective function to be optimized, and to the reweighting of violated global constraints defined by (12).

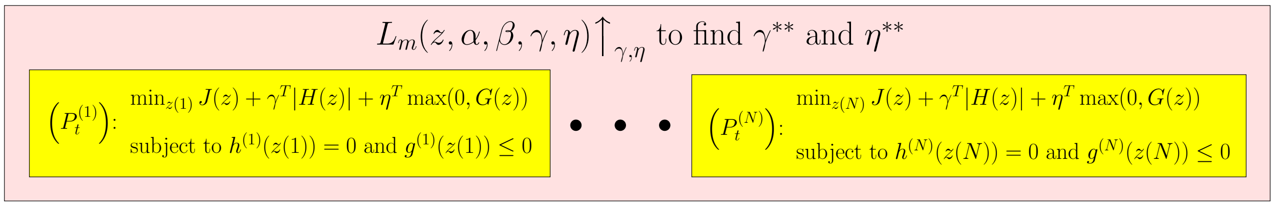
3.3 Formulation of Partitioned Planning Subproblems in PDDL2.2

For a PDDL2.2 planning problem solved in this paper, a solution plan is specified by the start time and the end time of each action $a \in \mathcal{O}$. Hence, its variable vector is $z = \{s(a), e(a) \text{ where } a \in \mathcal{O}\}$; its objective function $J(z)$ optimized depends on the makespan (or the number of actions for propositional domains) of plan z; and its constraints are the mutex constraints defined in Section 2.1:

$$h(a_i, a_j) = mutex\Big(s(a_i), e(a_i), s(a_j), e(a_j)\Big) = 0, \qquad \forall a_i, a_j \in \mathcal{O}. \quad (14)$$

Here, *mutex* is a binary procedure for checking whether a_i and a_j satisfy the mutex conditions defined in Section 2.1. It returns one if the conditions are satisfied and zero otherwise.

When the constraints are partitioned by their subgoals into N subproblems $G_1, \cdots, G_N$, variable z is partitioned into N subsets $z(1), \cdots, z(N)$, where $z(t)$ includes the start time


$L_m(z,\alpha,\beta,\gamma,\eta)\uparrow_{\gamma,\eta}$ to find γ^{**} and η^{**}
$\left(P_t^{(1)}\right)$: $\min_{z(1)} J(z)+\gamma^T|H(z)|+\eta^T\max(0,G(z))$ subject to $h^{(1)}(z(1))=0$ and $g^{(1)}(z(1))\le 0$
• • •
$\left(P_t^{(N)}\right)$: $\min_{z(N)} J(z)+\gamma^T|H(z)|+\eta^T\max(0,G(z))$ subject to $h^{(N)}(z(N))=0$ and $g^{(N)}(z(N))\le 0$

a) Partitioned search to look for points that satisfy (11) and (12)

```
1. procedure partition_and_resolve($P_t$)
2.     $\gamma \longrightarrow 0$; $\eta \longrightarrow 0$;
3.     repeat
           // increase the penalties of violated global constraints until maximum bounds $\bar{\gamma}_i$ and $\bar{\eta}_i$ //
4.         for $i = 1$ to $p$ do if ($H_i(z) \neq 0$ and $\gamma_i < \bar{\gamma}_i$) then increase $\gamma_i$ by $\delta$ end_if end_for;
5.         for $j = 1$ to $q$ do if ($G_j(z) \not\leq 0$ and $\eta_j < \bar{\eta}_j$) then increase $\eta_j$ by $\delta$ end_if end_for;
           // inner loop for solving the $N$ subproblems //
6.         for $t = 1$ to $N$ do apply an existing solver to solve (13) end_for;
7.     until (($\gamma_i > \bar{\gamma}_i$ for all $H_i(z) \neq 0$ and $\eta_j > \bar{\eta}_j$ for all $G_j(z) \not\leq 0$) or (a $CLM_m$ of $P_t$ is found))
8. end_procedure
```

b) Implementation for finding a CLM_m of P_t that satisfies (11) and (12)

Figure 7: The partition-and-resolve procedure to look for a CLM_m of P_t.

and the end time of those actions of G_t. The local constraints are those mutex constraints that relate the actions within a subproblem, and the global constraints are those that relate the actions across subproblems.

For $P_t^{(t)}$ defined for G_t, the objective is to find a feasible plan $z(t)$ that satisfies the constraints for G_t, while minimizing an objective function biased by the violated global constraints:

$$\left(P_t^{(t)}\right): \qquad \min_{z(t)} \quad J(z) + \sum_{\substack{k=1\\k\neq t}}^{N} \gamma_{t,k}\cdot m_{t,k} \tag{15}$$
$$\text{subject to} \qquad h^{(t)}(a_i,a_j)=0 \qquad \forall a_i,a_j \in z(t),$$

where $J(z)$ is defined later in Section 5.3. Here, $m_{t,k}$ is the number of global constraints between the actions in $z(t)$ and those in $z(k)$:

$$m_{t,k} = \sum_{\substack{a_t\in z(t)\\a_k\in z(k)\\k\neq t}} h(a_t,a_k). \tag{16}$$

To limit the number of penalties while characterizing the priorities among the subproblems, we have assigned a single penalty $\gamma_{t,k}$ for each pair of subproblems G_t and G_k, instead of a penalty for each global constraint between G_t and G_k.

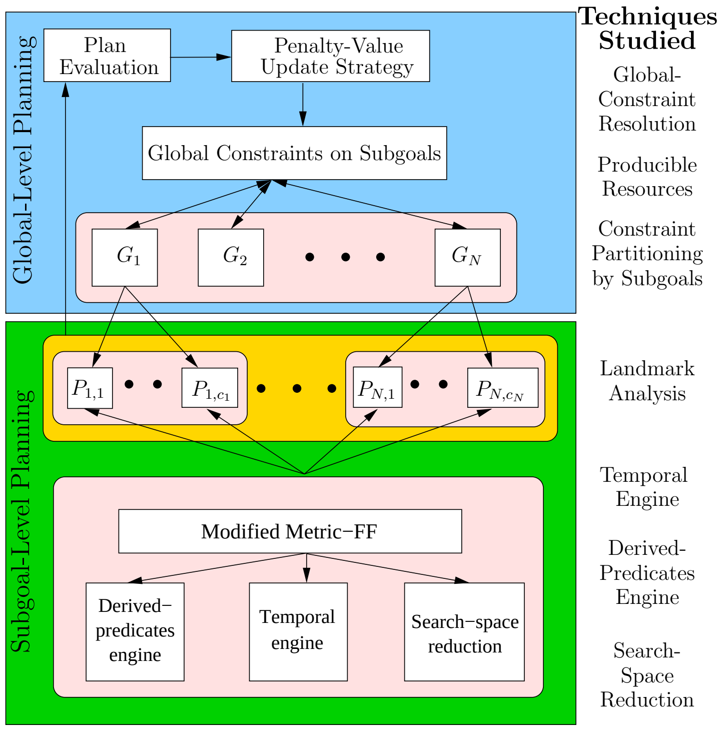


Figure 8: SGPlan$_4$: A planner implementing the partition-and-resolve procedure in Figure 7.

3.4 The Partition-and-Resolve Procedure

Figure 7 presents the *partition-and-resolve* procedure for finding points that satisfy the conditions in Theorem 2. Using fixed γ and η specified in the outer loop, the inner loop of Subproblem t in Figure 7b solves (13) by an existing solver, which results in an ESP that satisfies (11). This is possible because (13) is a well-defined MINLP. After solving the N subproblems, the penalties on the violated global constraints are increased in the outer loop. The process is repeated until a CLM_m to P_t has been found or when γ and η exceed their maximum bounds.

The procedure in Figure 7 may generate fixed points of (9) that do not satisfy (11) and (12). This happens because an ESP is a local minimum of (9) (but not the converse). One way to escape from such fixed points is to allow periodic decreases of γ and η. The goal of these decreases is to “lower” the barrier in the penalty function in order for local descents in the inner loop to escape from an infeasible region. Note that γ and η should be decreased gradually in order to help the search escape from such infeasible regions. Once γ and η reach their minimum thresholds, they can be scaled up, and the search is repeated.

4. System Architecture of SGPlan$_4$

Figure 8 shows the design of SGPlan$_4$ that implements the partition-and-resolve procedure. The procedure alternates between global-level planning and subgoal-level planning. In this section, we describe those techniques implemented in the global level, while leaving the discussion of techniques in the subgoal level to the next section.

4.1 The Partition-and-Resolve Process in SGPlan$_4$

At the global level, SGPlan$_4$ partitions a planning problem into N subproblems, $G_1, \cdots, G_N$, where G_t corresponds to the t^{th} subgoal. It then orders the subproblems, evaluates each using techniques in subgoal-level planning, identifies those violated global constraints, and updates their penalties in order to bias the search in the next iteration towards resolving them. In SGPlan$_4$, we have adopted an implementation in LPG1.2 (Gerevini & Serina, 2002) for detecting persistent mutexes.

The partition-and-resolve process can be understood as calculating subplans separately and then merging them into a consistent plan. Its goals are to optimize multiple subplans and to ensure their consistency after merging. Prior work on *plan merging* focuses on merging redundant actions and on finding an optimal composed plan. In particular, Foulser, Li, and Yang (1992) have developed algorithms for merging feasible classic plans into more efficient ones. A complete evaluation on plan-merging algorithms for classical domains has been conducted by Yang (1997). Tsamardinos, Pollack, and Horty (2000) have extended the concept to domains with temporal constraints. Because plan merging is not a means for making an infeasible plan feasible, it is different from our approach that aims to resolve inconsistencies in terms of mutexes among subplans.

An alternative view about our resolution approach is the reuse and modification of subplans into a consistent plan. Plan-reuse systems adapt existing plans into new initial states and goals. The approach is demonstrated in SPA (Hanks & Weld, 1995) and PRIAR (Kambhampati & Hendler, 1992) that show improvements in efficiency in many domains. The major difference between current plan-reuse approaches and our partition-and-resolve process is that we generate candidate subproblems based on the partitioning of mutex constraints, whereas traditional methods reuse plans that are generated by other means. Since the assumption of conservative plan modification in existing methods is not always achievable, it may be necessary to replan if a feasible plan candidate cannot be found. In some cases, it may be more expensive than planning from scratch. This is the reason why complexity analysis and empirical study cannot prove plan-reuse approaches to have consistent improvements over planing from scratch (Nebel & Koehler, 1995). In contrast, our approach augments the search of each subproblem by explicitly penalizing global inconsistencies and by forcing its solution towards resolving the global constraints.

Our partition-and-resolve approach is different from incremental planning (Koehler & Hoffmann, 2000) that uses a goal agenda. In incremental planning, a planner maintains a set of target facts, adds goal states incrementally into the target set, and extends the solution by using the new target set. Because a goal state must always be satisfied once it has been achieved, the ordering of goal states is important in order to avoid un-doing a previously achieved goal state when planning the current goal state. If invalidations do occur, then the planning task at that point is more complex than just the planning of one goal state. In contrast, SGPlan$_4$ tries to achieve only one subgoal at a time and allows other subgoals to be invalidated in the process. Moreover, for each subgoal, we do not need to start from the ending state of the previous subgoal as in incremental learning, and there is no need to pre-order the subgoals in order to avoid invalidations. We show in Section 6 that the performance of SGPlan$_4$ is not sensitive to the order of evaluating the subgoals.

4.2 Resolving Violated Global Constraints

In this section, we present two penalty-update strategies for resolving violated global constraints. These constraints are identified after finding a subplan for each subproblem independently.

SGPlan$_4$ first initializes the penalties of all global constraints when it starts. In the first iteration, SGPlan$_4$ solves each subproblem individually, without considering their global constraints. It then combines all the subplans into an integrated plan in order to determine the initial active global constraints across the subproblems. In subsequent iterations, SGPlan$_4$ finds a local feasible plan for each subproblem, while minimizing the global objective and the weighted sum of violated global constraints. At the end of each iteration, SGPlan$_4$ increases the penalty of a violated global constraint in proportion to its violation. The process ends when all the constraints are satisfied.

We have designed two strategies for updating the penalty of global constraints. The SGPlan$_4$ that participated in IPC4 sets very large initial penalty values and updates them by rate ξ, whereas SGPlan$_{4.1}$ studied in this paper sets the initial penalty values to zero:

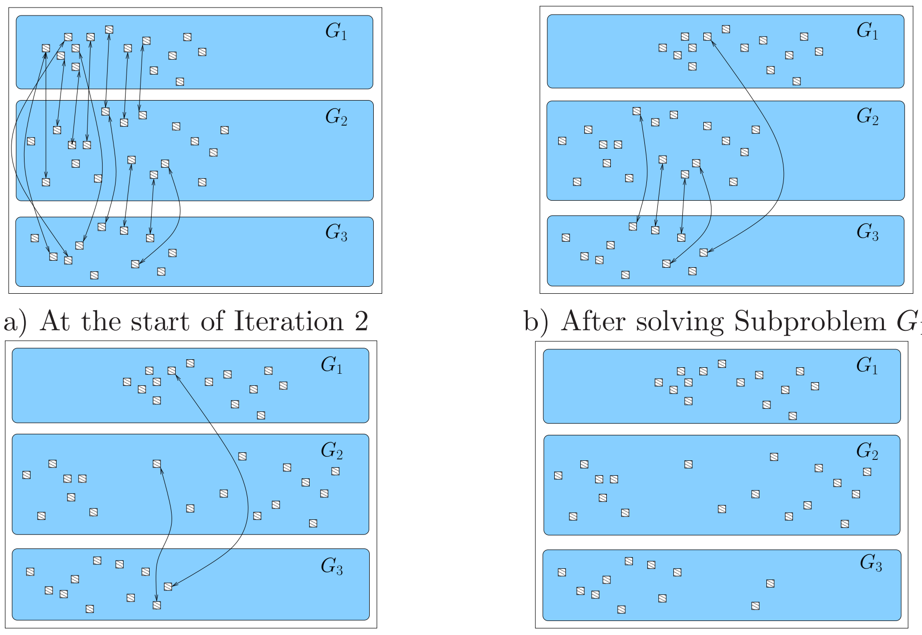
$$\gamma_{t,k}^{(0)} = \begin{cases} \gamma_0 & \text{(for SGPlan}_4\text{)} \\ 0 & \text{(for SGPlan}_{4.1}\text{)}, \end{cases} \qquad \gamma_{t,k}^{(\ell)} = \gamma_{t,k}^{(\ell-1)} + \xi \cdot m_{t,k}, \qquad \ell = 1, 2, \ldots \tag{17}$$

Here, $\gamma_{t,k}^{(\ell)}$ is the penalty for the global constraints between G_t and G_k in the ℓ^{th} iteration, $m_{t,k}$ is as defined in (16), γ_0 is a large initial value, and ξ is a parameter for controlling the rate of penalty updates. In our experiments, we set $\gamma_0 = 100$ and $\xi = 0.1$.

Figure 9 illustrates the planning process of SGPlan$_4$ on the AIRPORT-TEMP-14 instance. Given the three subproblems in this instance, SGPlan$_4$ first evaluates each subproblem once in the first iteration in order to determine the initial active global constraints. The figure shows, respectively, the subplans and the active global constraints after evaluating each of the three subproblems in the second iteration. The strategy is effective for reducing the number of active global constraints quickly from 14 in the beginning to zero in just one iteration.

The penalty-update strategy in SGPlan$_4$ may lead to longer makespans because it uses large initial penalty values in order to reduce the number of violated global constraints quickly. Hence, the subplans found may have poor temporal concurrency. To address this issue, we have implemented a new strategy for SGPlan$_{4.1}$ in (17) that sets the initial penalty values to zero.

Figure 10 illustrates the time-quality trade-offs of SGPlan$_4$ and SGPlan$_{4.1}$ when used to solve nine representative instances of the IPC4, the Blocksworld, and the Depots domains. Because the number of active global constraints changes after evaluating each subproblem, we plot the progress on the remaining number of active global constraints with respect to the total number of subproblems evaluated. The results show that both planners can



a) At the start of Iteration 2 b) After solving Subproblem G_1

c) After solving Subproblem G_2 d) After solving Subproblem G_3

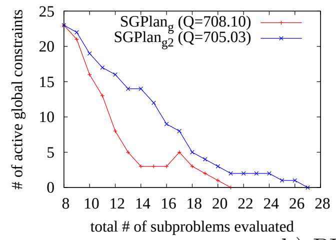
Figure 9: The planning process of the IPC4 version of SGPlan$_4$ in the second iteration in solving the AIRPORT-TEMP-14 instance. Each box corresponds to an action in a subplan, whereas each arrow corresponds to an active global constraint. By placing more emphasis on violated global constraints, the number of violated constraints is quickly reduced at the expense of a longer makespan.

resolve the remaining number of active global constraints in almost a linear fashion, and that SGPlan$_4$ is generally faster for resolving the active global constraints but generates plans of worse quality. In our detailed experimental results in Section 7, we show that SGPlan$_{4.1}$ generally leads to plans of better quality.

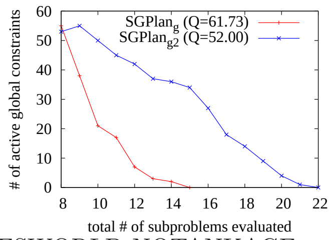
Both planners, however, have difficulty when solving the PIPESWORLD-NOTANKAGE-TEMP-DEADLINE-10 instance (Figure 10c). For this domain, SGPlan$_4$ cannot solve any instances, whereas SGPlan$_{4.1}$ can solve eight instances (1, 2, 5, 6, 8, 14, 22, and 30). Although the fraction of initial active global constraints out of all constraints is only 3.3% on average (Table 1), both planners may get stuck at some infeasible solutions and cannot make progress afterward. The reason is that the basic planner in both SGPlan$_4$ and SGPlan$_{4.1}$ does not have enough backtracking to generate new candidate subplans for each subproblem. Hence, the basic planner keeps generating the same subplan at some point, regardless of how the violated constraints are penalized.

4.3 Handling Producible Resources

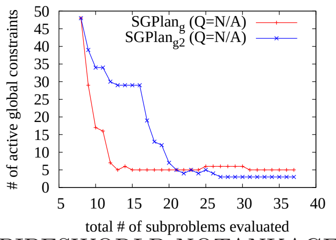
In some planning problems, there may be facts that can be made true and numerical resources that can be produced anytime when needed. For example, in the Settlers domain,



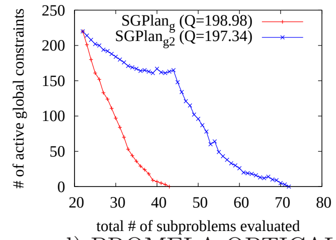
a) AIRPORT-TEMP-30



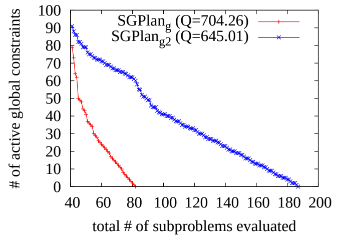
b) PIPESWORLD-NOTANKAGE-NONTEMP-30



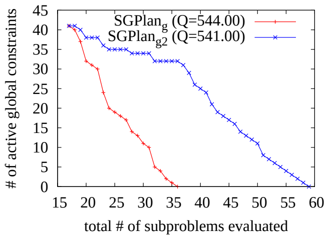
c) PIPESWORLD-NOTANKAGE-TEMP-DEADLINE-10



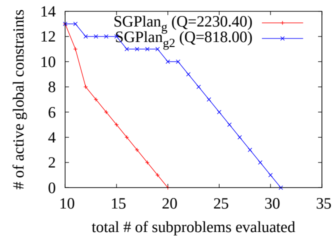
d) PROMELA-OPTICAL-TELEGRAPH-10



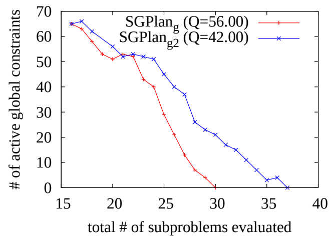
e) SATELLITE-TIME-20



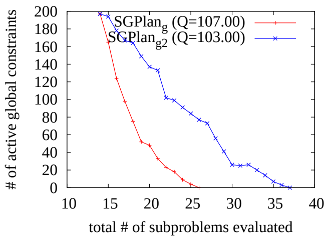
f) SETTLERS-20



g) UMTS-TEMP-50



h) BLOCKSWORLD-17-0



i) DEPOTS-TIME-20

Figure 10: Resolution of active global constraints in nine benchmark instances by the original penalty-update strategy in SGPlan$_4$ and the new penalty-update strategy in SGPlan$_{4.1}$. The x axis includes the number of subproblems evaluated, each corresponding to a subgoal, in the first iteration in order to determine the initial active global constraints.

coal can always be produced in a mine. We define these producible logical and numerical resources as follows.

a) A fact is producible if it is an add effect of either an action without preconditions or an action whose preconditions are always producible.

b) A numerical resource is producible if it is increased by either an action without preconditions or an action whose preconditions are always producible.

The planning tasks will be significantly easier if producible facts and resources can be detected in the preprocessing phase and be made available during planning. By first identifying all those facts and resources, SGPlan$_4$ derives a relaxed initial state by setting all producible facts to be true and all producible numerical resources to be large enough. Every time a producible fact is turned false, it is made true again. After finding a feasible plan from the relaxed initial state, SGPlan$_4$ removes the unused numerical resources in the initial state and plans again. The process is repeated until there are no redundant initial resources. At that point, SGPlan$_4$ inserts the necessary actions at the beginning of the plan to generate the minimum initial producible resources needed.

For example, suppose timber is detected to be a producible resource as one can always fell some trees to get more timber. SGPlan$_4$ will initially set a large number, say 1000 units, of timber available. After solving the problem, suppose there are 900 units left unused, it reduces the initial timber to 100 units and plans again. This process is repeated until either there is no unused timber at the final state or the problem becomes unsolvable after reducing the initial resource.

Note that the approach may incur some redundant actions for producing unused resources, as the optimal amount of resources needed cannot be predicted ahead of time.

5. Subgoal-Level Planning

At the subgoal level, SGPlan$_4$ applies landmark analysis to further partition a subproblem, performs path finding and optimization, carries out subspace-reduction analysis to prune irrelevant facts and actions in the subproblem, and calls a modified Metric-FF planner to solve the subproblem.

5.1 Subgoal-Level Decomposition Techniques

a) *Landmark analysis.* First proposed by Porteous, Sebastia, and Hoffmann (Porteous, Sebastia, & Hoffmann, 2001), landmark analysis allows a large planning problem to be decomposed into a series of simpler subproblems. Given the initial state, it aims to find some intermediate facts that must be true in any feasible plan for reaching the goal state. For example, assume that object O is to be delivered from A to D, and that the only path from A to D is $A \rightarrow B \rightarrow C \rightarrow D$. Then $AT(O, B)$ and $AT(O, C)$ are both landmark facts, since any feasible plan must make them true before reaching goal state $AT(O, D)$.

Because a planning problem is first partitioned by its subgoals into subproblems, we only apply landmark analysis on each subproblem in order to find the intermediate facts for reaching the corresponding subgoal. Landmark analysis is important in SGPlan$_4$ because it allows each subproblem to be further decomposed into simpler subproblems that can be solved easily.

In each subproblem, we find landmarks by a relaxed planning approach. Given a planning subproblem $\mathcal{T} = (\mathcal{O}, \mathcal{F}, \mathcal{I}, \mathcal{G})$, we first construct a relaxed planning graph from the initial state $\mathcal{I}$ by ignoring the delete effects of actions. We force each $f \in \mathcal{F}$ in each level of the graph to be false (even if it were made true by some actions). As a result, all the actions preconditioned by f will be pruned. If there exists a goal fact in $\mathcal{G}$ that cannot be reached when f is false, then f is a landmark fact and must be reached in any plan for the relaxed problem. After finding the partial order of the landmarks, SGPlan$_4$ builds a sequential list of subproblems joined by the landmarks found and applies the basic planner to solve each subproblem in order. Note that because landmark analysis is expensive, SGPlan$_4$ only detects landmarks once at the beginning and not in every iteration.

The landmarks found in the relaxed planning graph are necessary because any solution plan of the original problem is also a solution plan of the relaxed problem. Hence, any feasible plan for the original problem must reach each landmark found by the relaxed ap-

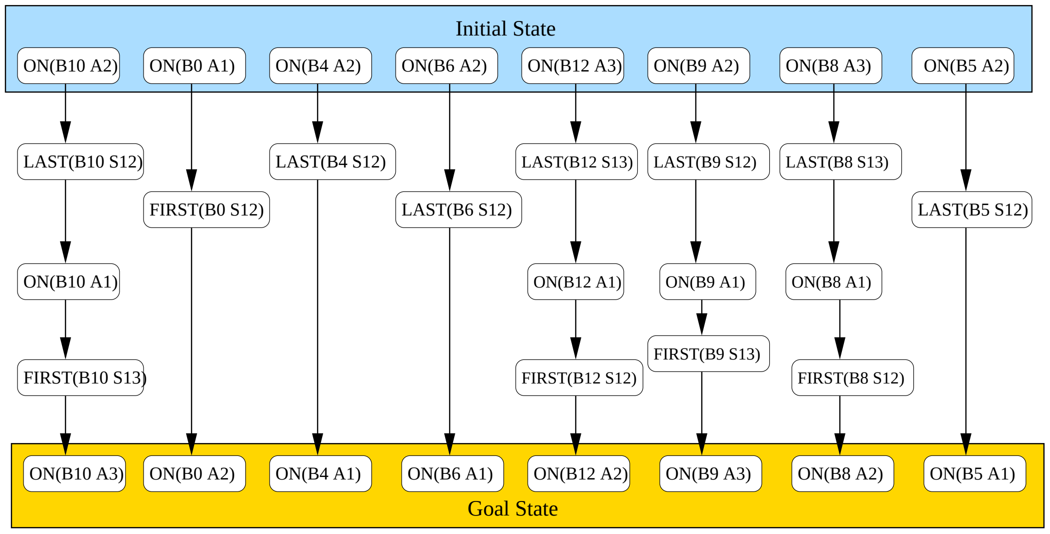


Figure 11: Landmarks and their partial orders for the PIPESWORLD-NOTANKAGE-NONTEMP-10 instance.

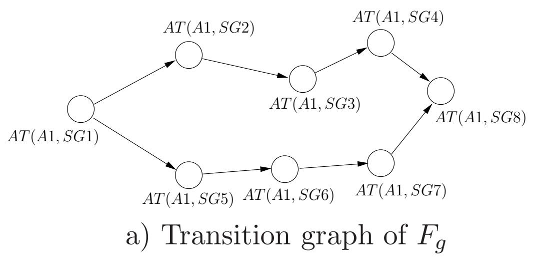
proach at least once. However, the landmarks found are not sufficient because we test goal reachability by a relaxed approach, and there may exist some undetected landmarks even when every fact has been tested.

Figure 11 shows all the landmarks found in the IPC4 PIPESWORLD-NOTANKAGE-NONTEMP-10 instance. When considering the first goal fact $ON(B10, A3)$, $LAST(B10, S12)$ is not only its landmark but also the landmarks for $ON(B10, A1)$ and $FIRST(B10, S13)$. This means that $LAST(B10, S12)$ must be ordered before $ON(B10, A1)$ and $FIRST(B10, S13)$. In this way, we can decompose the subproblem for $ON(B10, A3)$ into 4 smaller tasks that must be carried out in sequence, namely, $LAST(B10, S12)$, $ON(B10, A1)$, $FIRST(B10, S13)$, and $ON(B10, A3)$.

b) *Landmarks identified by path finding.* Landmark analysis may sometimes produce very few landmark facts for decomposing a subproblem. For example, most of the gates along a path in an Airport instance will not be identified as landmark facts (that is, must-visit points) because there are usually multiple paths for the given source and destination. Consider the airport topology in Figure 12a in which the goal is to move $A1$ from $SG1$ to $SG8$. Because there are two alternative paths and none of the facts in $AT(A1, SG2), AT(A1, SG3), \cdots, AT(A1, SG7)$ has to be true before reaching $SG8$, we cannot detect any landmark facts.

To identify more landmark facts for decomposing a subproblem, we have developed in SGPlan$_4$ a new path-finding technique. The technique is based on the concept of fact groups that has been used by some existing planners, such as MIPS (Edelkamp, 2002) and Downward (Helmert & Richter, 2004). A fact group includes a group of mutually exclusive facts in which only one can be true at any time, and typically involves the multiple possible states of an object. For the example Airport instance discussed above, a fact group includes the different locations that $A1$ can be at:

$$F_g = \Big\{AT(A1, SG1), AT(A1, SG2), \cdots, AT(A1, SG7), AT(A1, SG8)\Big\}. \tag{18}$$



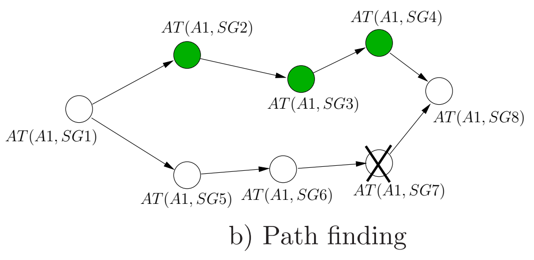


Figure 12: Illustration of the transition graph of Fact Group F_g and the path finding algorithm. Shaded nodes in (b) are new landmark facts detected by path finding.

In SGPlan$_4$, we have adopted an approach in MIPS based on an analysis of static mutex groups for finding fact groups of subgoal facts.

We apply path finding on Subproblem G_t when none or a few landmarks have been detected by landmark analysis. Assuming the subgoal to be reached is g_t, we first find the fact group it belongs to. In the previous example, the subgoal is $g_t = AT(A1, G8)$, and the fact group is F_g in (18).

For each fact group with two or more facts, we determine their transition relations by constructing a directed graph. Given two facts f_1 and f_2 in the fact group, we add an edge from f_1 to f_2 if there exists an action a such that f_1 is a precondition of a and f_2 is an add effect of a (which implies that f_1 is a delete effect of a since f_1 and f_2 are mutually exclusive). Figure 12a illustrates the transition graph for the airport example discussed above.

Last, to find a path, we look for all the facts that are immediate predecessors of g_t in the graph. We arbitrarily select one as a must-visit landmark and disable the others. We then perform a landmark analysis from the initial fact to g_t. This analysis will return more landmark facts.

In our example airport instance, $AT(A1, SG4)$ and $AT(A1, SG7)$ are the two immediate predecessor facts of Subgoal $g_t = AT(A1, SG8)$. If we disable $AT(A1, SG7)$ in the landmark analysis, then there will only be one path from $AT(A1, SG1)$ to g_t, and $AT(A1, SG2)$, $AT(A1, SG3)$, and $AT(A1, SG4)$ will be detected as landmark facts. Figure 12b illustrates this process.

c) *Path optimization* is used to find better landmark facts for problems with timed initial literals or numerical effects. It is invoked when there is a deadline or when there is a dynamically changing numerical resource that appears in the preconditions of actions. These conditions are satisfied in the IPC4 Satellite instances where the technique is found to be most useful.

The technique works by choosing a path that optimizes the time duration or the usage of a numerical resource when there are multiple paths of different quality, and by setting those nodes along the optimal path as landmark facts. Given a subproblem trying to reach Subgoal g_t, we construct a transition graph for the fact group of g_t and apply Dijkstra's algorithm to find the shortest path from the initial fact to g_t. The weight on each edge is either a time duration for problems with time windows, or the usage of a numerical resource

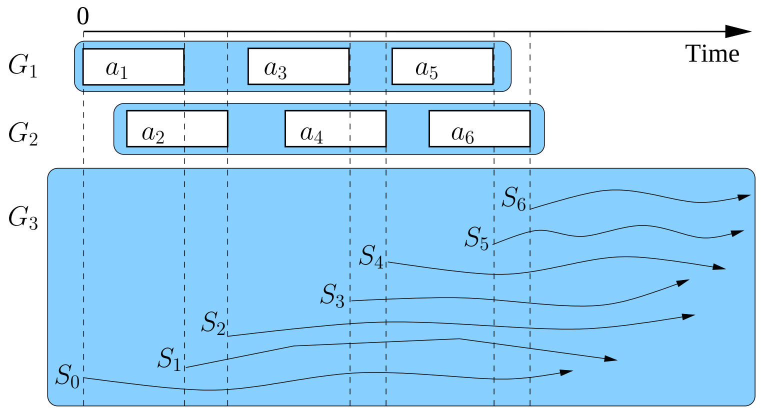


Figure 13: Generating multiple starting states for Subproblem G_3, given the initial state S_0 and $S_i, i = 1, \ldots, 6$, the state when action a_i is finished. SGPlan$_4$ calls the basic planner to generate a local subplan from each starting state and picks the first one that improves the objective in (15).

for problems with numerical preconditions. We then set the facts along the optimal path as landmark facts and force the planner to choose this path over others. The landmarks along the optimal path allows us to further decompose the problem into subproblems.

There are two limitations in our current implementation of path optimization. First, since there needs to be a path from the initial fact to the goal fact in the transition graph, we cannot apply the technique if the initial and the goals facts are disconnected. Second, we have studied the case of only one dynamically changing numerical resource that appears in the preconditions of actions and have not studied the optimization of multiple numerical resources.

5.2 Subgoal-Level Planning Techniques

a) *Evaluating multiple subplans for a subproblem.* In finding a local feasible subplan for a subproblem that improves the objective in (15), SGPlan$_4$ generates a number of subplans from multiple starting states. Since no active global constraints exist between two identical subplans, we generate multiple starting states for a given subproblem by applying all possible prefix actions from each of the other subproblems. For example, given the six actions planned in G_1 and G_2 in Figure 13, there are six possible starting states when developing a subplan for G_3. For each starting state, SGPlan$_4$ calls the basic planner to generate a local feasible subplan and accepts the subplan if it improves the objective in (15). If no better subplans can be found from all possible starting states, SGPlan$_4$ leaves the local subplan unchanged and moves on to the next subproblem.

b) *Search-space reduction.* Before solving a partitioned subproblem, we can often eliminate in its search space many irrelevant actions that are related to only facts and subgoals in other subproblems. Such reductions are not useful in planning problems that are not partitioned because all their actions are generally relevant.

As an example, consider a transportation domain whose goal is to move packages, drivers, and trucks to various locations from an initial configuration. Suppose in a problem instance, the goal set is $\{AT(D1, S1),\ AT(T1, S1),\ AT(P1, S0),\ AT(P2, S0)\}$ for two packages $P1$ and $P2$, one driver $D1$, one truck $T1$, and two locations $S1$ and $S2$. Without partitioning, all the actions are relevant for resolving the subgoals. In contrast, after partitioning, the actions for moving $P2$ around are irrelevant in the subproblem of resolving $AT(P1, S0)$ and can be eliminated. Similarly, those actions for moving $P1$ or $P2$ are irrelevant in the subproblem of resolving $AT(D1, S1)$.

We have designed a *backward relevance analysis* to eliminate some irrelevant actions in a subproblem before solving it by the basic planner. In the analysis, we maintain an *open list* of unsupported facts, a *close list* of relevant facts, and a *relevance list* of relevant actions. In the beginning, the open list contains only the subgoal facts of the subproblem, and the relevance list is empty. In each iteration, for each fact in the open list, we find all the actions that support the fact and not already in the relevance list. We then add these actions to the relevance list and add the action preconditions that are not in the close list to the open list. We move a fact from the open list to the close list when it is processed. The analysis ends when the open list is empty. At that point, the relevance list will contain all possible relevant actions. This analysis takes polynomial time.

Note that our relevance analysis is not complete when it stops, since the relevance list may still contain some irrelevant actions. For example, we can further reduce the relevance list by a forward analysis and by finding all applicable actions from the initial states before the backward analysis. However, further analysis may not be cost effective for reducing the overhead in planning.

Our reduction method belongs to a family of heuristics proposed by Nebel, Dimopoulos and Koehler (1997). Since we select all possible supporting actions when processing a fact, our approach is indeed the one that selects the union over all elements in the possibility set according to their classification. While we conservatively reduce the irrelevant information, there are a number of tighter reductions that can approximately minimize the use of initial facts (Nebel et al., 1997). However, these aggressive heuristics may not be solution preserving or solution-length preserving.

5.3 Modified Metric-FF Basic Planner

After decomposing a subproblem associated with a subgoal into smaller subproblems bounded by landmark facts, SGPlan$_4$ solves each subproblem identified (or the original subproblem in case no landmark facts have been identified) by a modified Metric-FF planner. Our modifications consist of two components: the adaptation of the original Metric-FF (Hoffmann, 2003) in order to entertain the new features in PDDL2.2, and the support of planning when the mutex constraints are partitioned. In fact, a lot of our efforts for embedding Metric-FF in SGPlan$_4$ were spent on the first component.

The original Metric-FF can only solve problems in PDDL2.1 with propositional actions but does not support any temporal features. We have extended the parser of Metric-FF to support the full PDDL2.2 syntax and the definition of actions from atomic logical to durational temporal. The planning process has also been extended from sequential propositional planning to parallel temporal planning. Specifically, we have extended sequential actions of

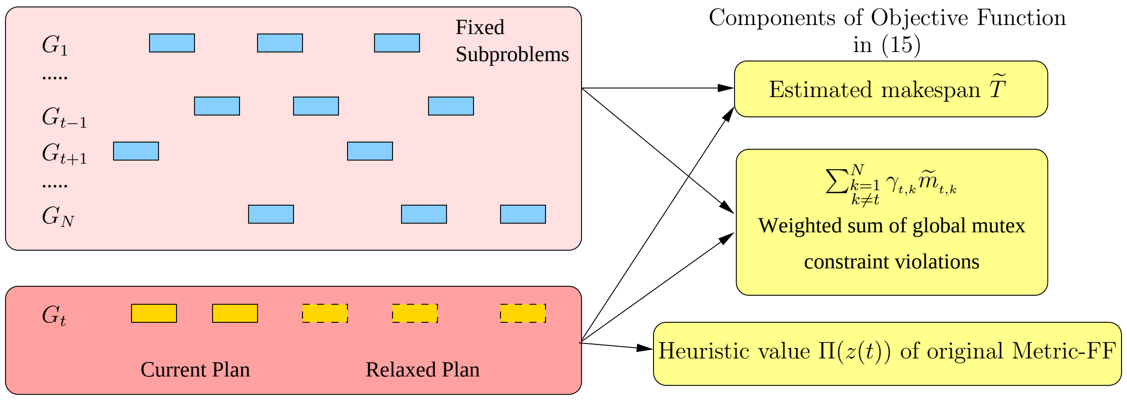

G_1
.....
G_{t-1}
G_{t+1}
.....
G_N
Fixed Subproblems
G_t
Current Plan
Relaxed Plan
Components of Objective Function in (15)
Estimated makespan $\widetilde{T}$
$\sum_{k=1, k\neq t}^{N} \gamma_{t,k}\widetilde{m}_{t,k}$
Weighted sum of global mutex constraint violations
Heuristic value $\Pi(z(t))$ of original Metric-FF

Figure 14: Temporal planning in a partitioned search context incorporates in the objective function in (15) a makespan $\widetilde{T}$ estimated by an enhanced PERT algorithm and the heuristic value of the Metric-FF planner.

atomic length in the original Metric-FF to actions with predefined durations that can be scheduled in parallel.

We have extended Metric-FF to support a new feature called derived predicates introduced in PDDL2.2. Derived predicates define axioms whose facts are derived by a set of precondition facts. For example, in a domain with boxes, if A is above B and B is above C, then a derived predicate of A above C can be generated. Derived predicates can only appear in preconditions and goals but not in effects. In our modified Metric-FF, we have implemented a technique proposed in MIPS 2.2 (Edelkamp, 2003) for handling derived predicates. We encode any derived predicate d as a special action a, where the precondition facts of a are the preconditions facts of d, the add effects of a are the derived facts of d, and the delete effect of a is empty. During planning, all the "derived-predicate actions" are included in the relaxed plan. However, the heuristic function computed in Metric-FF only counts the number of real actions in the relaxed plan but not the number of "derived-predicate actions," and only real actions are considered as candidates for forward expansion in any state. In any state, we expand the set of true facts by applying all applicable derived predicates iteratively until we reach a fixed-point state where no more true facts can be added.

The second component of the modifications in Metric-FF involves the support of a partitioned search context when solving a subproblem, say G_t. In this case, Metric-FF needs to incorporate in its objective an aggregate state of all schedulable actions in $G_1, \cdots, G_N$ in the planning of actions in G_t. Referring to Figure 14, the aggregate state is represented by an estimated makespan $\widetilde{T}$ of all the actions that is evaluated by an enhanced PERT algorithm.

PERT was originally developed to generate a parallel plan by scheduling an action as early as possible until it is blocked by a dependency or a mutex relation. Previous PERT algorithms detect a propositional conflict between two actions by checking if one action adds/deletes another's precondition, and detect a numerical conflict when two actions modify the same numerical variable. In the latter case, two actions would not be allowed to overlap in their execution when they consume the same resource, even when the total amount required does not exceed the amount available. Obviously, the resulting schedule will be suboptimal.

We have developed an enhanced PERT algorithm that considers resource constraints in its schedule. The algorithm assigns an action as early as possible as long as there are no propositional conflicts or no violations on numerical/resource constraints. Besides maintaining operator dependency as in the original PERT, we also keep track of changes on numerical variables. Our algorithm is greedy because it schedules all applicable actions as early as possible without backtracking.

In general, PERT can schedule a valid sequential plan into a parallel plan without mutex conflicts. However, our enhanced PERT may generate a parallel plan with mutex conflicts. The reason is that each subproblem is solved from the initial state and not sequentially from the state of the previous subproblem. Hence, when actions from multiple subplans are combined, one action may delete the precondition of another and causes a mutex conflict. As an example, consider the sequential plans of two subproblems G_1 and G_2 that are scheduled from the initial state in the Blocksworld domain: a) $MOVE(A, B)$ and $MOVE(B, C)$; and b) $MOVE(D, E)$ and $MOVE(E, C)$, where $MOVE(x, y)$ places x on top of y, with a precondition $CLEAR(y)$ (y is clear with nothing on it). In this example, PERT cannot generate a parallel plan with no mutex conflict between $MOVE(E, C)$ and $MOVE(B, C)$, regardless of how these two actions are scheduled. The conflict occurs because each action deletes the $CLEAR(C)$ precondition of the other.

The modified Metric-FF planner carries out a search that heuristically looks for plans to minimize (15) rewritten as follows:

$$\min_{z(t)} \left(J(z) + \sum_{\substack{j=1 \\ j \neq t}}^{N} \gamma_{t,j} \widetilde{m}_{t,j} \right) = \begin{cases} \min_{z(t)} \left(\Pi(z(t)) + \sum_{\substack{k=1 \\ k \neq t}}^{N} \gamma_{t,k} \widetilde{m}_{t,k} \right) & \text{(for SGPlan}_4\text{)} \\ \min_{z(t)} \left(\Pi(z(t)) + \tau \widetilde{T} + \sum_{\substack{k=1 \\ k \neq t}}^{N} \gamma_{t,k} \widetilde{m}_{t,k} \right) & \text{(for SGPlan}_{4.1}\text{)}, \end{cases} \tag{19}$$

where $\Pi(z(t))$ is the heuristic value of the original Metric-FF when solving G_t; $\widetilde{m}_{t,k}$ is the estimated number of active mutexes between the plan for G_k and a relaxed plan for G_t obtained by ignoring the delete effects of unscheduled actions; $\widetilde{T}$ is the makespan estimated by the enhanced PERT algorithm after composing the relaxed plan of G_t and the plans of the other subproblems; $\gamma_{t,k}$ is a penalty value dynamically updated in global-level planning; and τ is a constant fixed at 0.0001. Although the search does not guarantee optimality, it can always resolve global mutual-exclusion constraints between, say $z(t)$ and $z(k)$, because it can move one subplan backward in order to avoid overlapping with another conflicting subplan when the penalty $\gamma_{t,k}$ is large enough.

In our implementation of (19) in the modified Metric-FF planner, we have set τ in SGPlan$_{4.1}$ to be very small so that the penalty term due to the makespan will not dominate the other terms. In fact, since $\tau\widetilde{T}$ is much smaller than one in all the test problems, its main purpose is to break ties among those states with very close heuristic values. On the other hand, our implementation of (19) in SGPlan$_4$ in IPC4 does not include $\widetilde{T}$ in its objective function. As a result, it focuses on eliminating mutual-exclusion conflicts and tends to generate plans of a longer makespan.

```
1.  procedure SGPlan(problem file)
2.      parse problem file and instantiate all facts and actions;
3.      detect and encode timed initial literals (TIL);
4.      detect and encode derived predicates;
5.      detect TIL wrappers and translate them into regular TILs;
6.      detect producible resources;
7.      if (there are producible resources) then set them to the maximum possible end_if;
8.      repeat
9.          for each subgoal fact in the goal list do
10.             call search-space reduction to eliminate irrelevant actions;
11.             call basic planner (modified Metric-FF) to reach the subgoal;
12.             if (the basic planner times out) then
13.                 perform landmark analysis to generate a list of subproblems;
14.                 for each subproblem in the list do
15.                     call basic planner to solve the subproblem;
16.                     if (solution is not found in the time limit) then
17.                         if (problem has TIL or numerical fluents) then perform path optimization
18.                         else perform path finding to further decompose the subproblem end_if;
19.                         call basic planner to solve each decomposed subproblem;
20.                     end_if
21.                 end_for
22.             end_if
23.         end_for
24.         evaluate plan $z$ and update penalty values of violated global constraints;
25.     until feasible solution plan has been found or time limit has been exceeded;
26.     if ((new solution found) && (there are unused producible resources)) then
27.         reduce the initial producible resources and goto step 8;
28.     end_if
29. end_procedure
```

Figure 15: The high-level pseudo code common for both SGPlan$_4$ and SGPlan$_{4.1}$.

In general, embedding a basic planner in our partition-and-resolve framework requires some modifications to the objective function of the basic planner in order to implement (15). Hence, it cannot be done without the source code of the basic planner.

5.4 Putting All the Pieces Together

Figure 15 shows the high-level code that is common for both SGPlan$_4$ and SGPlan$_{4.1}$. The preprocessing phase parses the problem file and instantiates all the facts and actions (Line 2), detects and encodes timed initial literals (TIL) and derived predicates, if any (Lines 3 and 4), translates the problem into a regular TIL problem if the problem is a compiled TIL problem (Line 5), and detects producible resources and sets them to always available (Lines 6 and 7).

The major loop is between Lines 8 and 28. For each subgoal, SGPlan$_4$ uses search-space reduction to eliminate irrelevant actions (Line 10) and solves it using the basic planner (Line 11). If the basic planner fails to find a feasible plan within a time limit (3000 node

Table 2: Summary of useful techniques for each domain variant. A check mark indicates that a technique is found to be useful for a domain variant or a class of domain variants.

Domain Variant	SG	LM	PF	PO	TIL	TIL-w	DP	PR	SR
AIRPORT-*	✓	✓	✓						
AIRPORT-TEMP-TIMEWINDOWS-CO	✓	✓	✓			✓			
PIPESWORLD-*	✓	✓	✓						✓
PROMELA-*	✓								
PROMELA-*-DP	✓						✓		
PSR-SMALL	✓								
PSR-MIDDLE	✓						✓		✓
PSR-MIDDLE-CO	✓								✓
PSR-LARGE	✓								✓
SATELLITE-STRIPS	✓								
SATELLITE-TIME	✓								
SATELLITE-NUMERIC	✓	✓		✓					
SATELLITE-COMPLEX	✓								
SATELLITE-TIME-TIMEWINDOWS	✓	✓		✓	✓				
SATELLITE-TIME-TIMEWINDOWS-CO	✓	✓		✓		✓			
SETTLERS	✓							✓	
UMTS-TEMP	✓								
UMTS-TEMP-TIMEWINDOWS	✓				✓				
UMTS-TEMP-TIMEWINDOWS-CO	✓					✓			
UMTS-FLAW-TEMP	✓								
UMTS-FLAW-TEMP-TIMEWINDOWS	✓				✓				
UMTS-FLAW-TEMP-TIMEWINDOWS-CO	✓					✓			

Keys						
	SG:	subgoal partitioning	LM:	landmark analysis	PF:	path finding
	PO:	path optimization	TIL:	timed initial literals handling	TIL-w:	TIL wrapper detection
	DP:	derived predicates handling	PR:	producible resources	SR:	search-space reduction

expansions in Metric-FF), SGPlan$_4$ aborts the run of Metric-FF and tries to decompose the problem further. It first applies landmark analysis to decompose and solve the subproblem (Lines 13-15). If it is unsuccessful in solving the subproblem, it tries path optimization for numerical and TIL problems (Line 17) or path finding (Line 18) to further partition the subproblem. After all the subgoals have been evaluated, it composes the solution, evaluates the global constraints, and updates the penalty values (Line 24). Finally, if a new solution has been found and there are unused producible resources, it reduces the initial producible resources (Lines 26-28) and repeats the major loop again.

6. Sensitivity Analysis of Techniques in SGPlan$_4$

In this section we describe our ablation study of the various techniques in SGPlan$_4$ in order to test their effectiveness. Table 2 lists the techniques that are most useful for each IPC4 domain variant. We defer the discussion on the performance improvement of SGPlan$_{4.1}$ over SGPlan$_4$ to Section 7.

For all the Airport variants, the useful techniques include subgoal partitioning, landmark analysis, and path finding. In addition, TIL wrapper detection is needed for the TIMEWINDOWS-CO variant. As an ablation study, we applied SGPlan$_4$ with subgoal partitioning alone. In this case, SGPlan$_4$ can solve 107 out of the 200 (53.5%) instances and cannot solve those numbered higher than 28 (namely, P29, P30, etc.). The reason is that those subproblems without landmark analysis and path finding are so large that Metric-FF has difficulty in solving them. In contrast, SGPlan$_4$ with landmark analysis and path finding can solve 159 (79.5%) instances.

For all the Pipesworld variants, the useful techniques include subgoal partitioning, landmark analysis, path finding, and search-space reduction. Although search-space reduction can slightly reduce the run time by 5.3% on average, landmark analysis and path finding has more significant effects on performance. SGPlan$_4$ without landmark analysis and path finding can only solve 102 out of the 200 (51%) instances, whereas SGPlan$_4$ with landmark analysis and path finding can solve 186 instances (93%). Landmark analysis and path finding also leads to 8% average improvement on run time for those instances that both versions can solve.

For the Promela domain, only subgoal partitioning is found to be useful, besides applying derived-predicate handling for the corresponding variants.

For all the PSR variants except PSR-SMALL, search-space reduction is particularly useful in addition to subgoal partitioning. For these three variants, SGPlan$_4$ with search-space reduction can solve, respectively, 50, 14, and 11 instances; whereas SGPlan$_4$ without search-space reduction can solve, respectively, 47, 8, and 6 instances. In addition, the average run-time improvements due to search-space reduction are, respectively, 34.1%, 46.9%, 62.5%. For the PSR-SMALL variant, search-space reduction has no significant effects on both run time and solution quality. Last, derived-predicate handling is important for PSR-MIDDLE, which is encoded using derived predicates.

In the Satellite domain, only subgoal partitioning is found to be useful for solving the TIME, STRIPS, and COMPLEX variants. For the NUMERIC, TIME-TIMEWINDOWS, and TIME-TIMEWINDOWS-CO variants, landmark analysis and path optimization are also useful. For these three variants, SGPlan$_4$ can solve, respectively, 25, 25, and, 21 instances, whereas SGPlan$_4$ without landmark analysis and path optimization can solve, respectively, 16, 16, and 13 instances.

For the Settlers domain, subgoal partitioning as well as techniques for handling producible resources are important for solving all but one of the instances. (The eighth instance is infeasible.) Without detecting producible resources, SGPlan$_4$ can only solve nine out of the 20 instances.

For the UMTS domain, only subgoal partitioning is found to be useful, besides applying TIL handling and TIL wrapper detection for the corresponding variants. Landmark analysis does not help in this domain and can detect none or very few landmark facts in each of the 300 instances. Also, search-space reduction can only prune a few facts and has little effects on performance.

We have also studied the effects of subgoal ordering in SGPlan$_{4.1}$ on eighteen representative variants from all IPC4 domains as well as the Depots domain (Figure 16). For each instance, we test SGPlan$_{4.1}$ using five random subgoal orders and normalize its run time (*resp.* quality) with respect to the corresponding measure when SGPlan$_{4.1}$ is run using

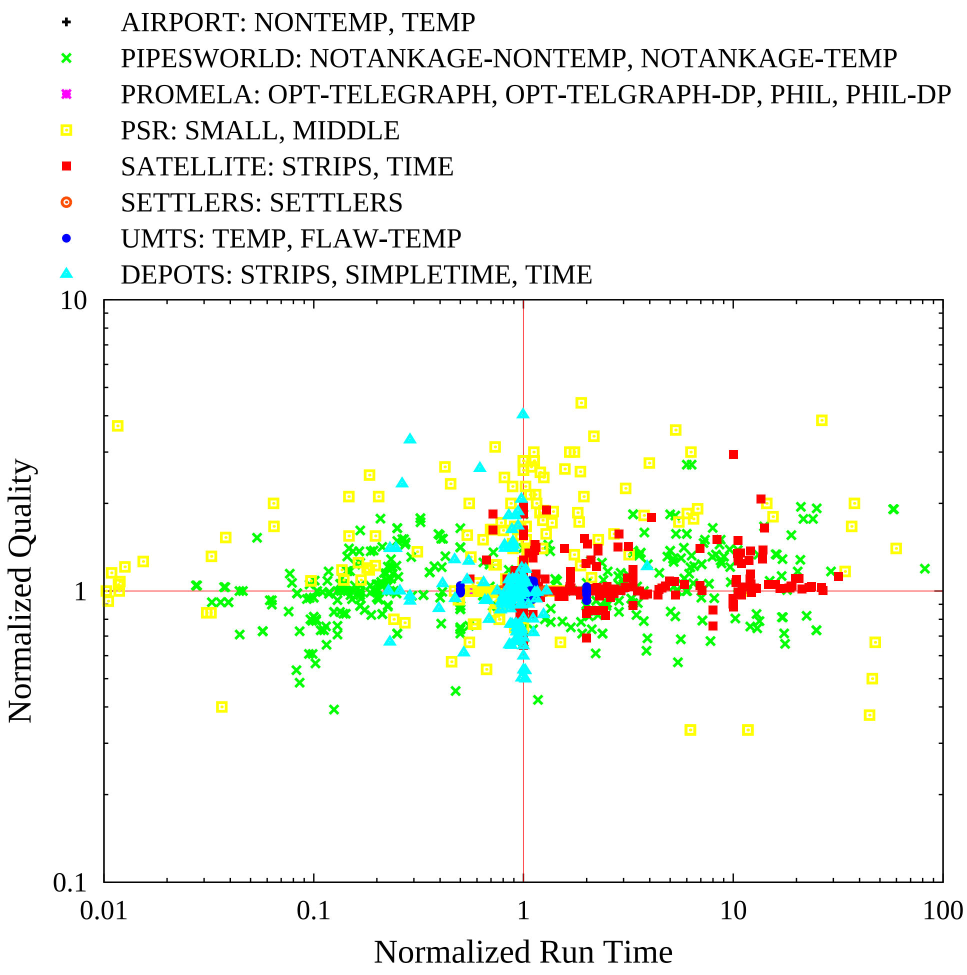


Figure 16: Run time-quality distribution of $\text{SGPlan}_{4.1}$ run using different random subgoal orders on selected IPC4 and the Depots domain variants. The results are normalized with respect to the run time and quality of $\text{SGPlan}_{4.1}$ run using the default subgoal order. (Performance values larger than one are better for $\text{SGPlan}_{4.1}$.)

the original order in the problem definition. Here we use makespan as our quality measure for temporal domains and the number of actions for propositional domains (even when an objective is specified in the problem definition).

The results show that the performance of $\text{SGPlan}_{4.1}$ is quite insensitive to subgoal ordering for the Airport, Promela, Settlers, and UMTS domains. However, there are significant variations in run time and quality for the Pipesworld and PSR domains, although there is no definitive trend that a random subgoal order is better. For the Depots domain, there exist some smaller variations in both run time and quality. A common feature among the Pipesworld, PSR, and Depots domains is that they all have intensive subgoal interactions, which make them more sensitive to the order in which subgoals are evaluated. For example, in the PSR-MIDDLE variant, the number of subgoals is large, and different subgoals are highly related by derived predicates. Last, we note that using the original subgoal order leads to better run time and quality in the Satellite domain. The reason is that the original order can avoid unnecessary subgoal invalidations when finding local feasible subplans, since the starting states are generated by applying prefix subplans of other subgoals.

Because there is no clear advantage of using random subgoal orders over the original subgoal order, SGPlan$_4$ and SGPlan$_{4.1}$ use the original subgoal order in their implementations.

7. Experimental Results

In this section, we experimentally compare the performance of SGPlan$_4$, SGPlan$_{4.1}$ (their differences are in (17) and (19)) and other planners in solving the IPC3 and IPC4 benchmark suites as well as the Blocksworld domain from IPC2. Each suite contains multiple domains, with several variants in each. Those variants in IPC4 address the different features of PDDL2.2, which include versions on STRIPS, STRIPS with DP (derived predicates), temporal, temporal with TIL (deadlines), numeric, and complex (temporal and numeric). A complete description of each variant and its problem files can be found at the Web site of each of the competitions[2]

All runs were carried out an AMD Athlon MP2800 PC with Redhat Linux AS3 and 2-Gbyte main memory unless otherwise noted. Following the rules of IPC4, all random planners set a fixed random seed, once and for all, throughout their experiments. Moreover, all planners must be fully automated, run with the same parameter setting for all the instances attempted, and execute under a CPU time limit of 30 minutes and a main memory limit of 1 Gbytes.

Table 3 summarizes the performance of SGPlan$_4$, SGPlan$_{4.1}$, Downward (Helmert & Richter, 2004), LPG-TD-SPEED-1.0 with a seed of 2004, and YAHSP-1.1.[3] We use makespan as the quality metric for temporal domains and the number of actions for propositional domains. Since the code for Downward is unavailable, we report its IPC4 results after adjusting its run times by a factor governed by the difference in speeds between the computer used in the IPC4 competition and the computer used for SGPlan$_{4.1}$. Likewise, we were unable to evaluate Downward on the IPC2 and IPC3 benchmarks.

Table 3 does not include results on those domain variants that a target planner cannot handle. For example, LPG-TD-SPEED cannot solve all the compiled domains and does not support some grammatical features in PSR-LARGE and the two FLUENTS variants in the PROMELA domain; and YAHSP cannot handle derived predicates. In contrast, both SGPlan$_4$ and SGPlan$_{4.1}$ were designed to solve all the variants except the ROVERS-TIME variant with dynamic durations. Note that since the Satellite and the Settlers domains exist in both the IPC3 and IPC4 benchmarks, the table does not include those results on

2. The URL for the competitions are `http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/` for IPC4, `http://planning.cis.strath.ac.uk/competition/` for IPC3, and `http://www.cs.toronto.edu/aips2000/` for IPC2.
3. The object code of LPG-TD was downloaded from `http://zeus.ing.unibs.it/lpg/register-lpg-td.html`, while the object code of YAHSP-1.1 was downloaded from `http://www.cril.univ-artois.fr/~vidal/Yahsp/yahsp.linux.x86.gz`. The object code of Downward was unavailable for testing at the time when this paper was revised.

the IPC3 Settlers domain and some variants of the IPC3 Satellite domain that have been reported for IPC4.

Table 3: Performance comparison between SGPlan$_{4.1}$ and other planners. In the table comparing SGPlan$_{4.1}$ and SGPlan$_4$, the four missing variants (PIPESWORLD-NOTANKAGE-TEMP-DEADLINES-CO, PROMELA-OPTICAL-TELEGRAPH-FLUENTS-DP, PROMELA-PHILOSOPHERS-FLUENTS-DP, and ROVERS-TIME) cannot be solved by both planners. In the table comparing SGPlan$_{4.1}$ and LPG-TD-SPEED, all the missing variants except ROVERS-TIME cannot be solved by LPG-TD-SPEED. For the ROVERS-TIME variant, only LPG-TD-SPEED can solve all the instances but the other planners cannot. In the tables comparing SGPlan$_{4.1}$, Downward, and YAHSP, all the missing variants cannot be solved by the target planners compared.

Domain Variant	Instances Solvable by Both (F_b)						All Instances			
	F_i	F_q	F_t	F_w	F_{wt}	F_{wq}	F_n	F_g	F_u	F_b
Comparison between SGPlan$_{4.1}$ and SGPlan$_4$										
AIRPORT-NONTEMP	0.78	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.12	0.88
AIRPORT-TEMP	0.60	0.28	0.00	0.00	0.00	0.00	0.00	0.00	0.12	0.88
AIRPORT-TEMP-TIMEWINDOWS	0.48	0.14	0.16	0.06	0.00	0.02	0.00	0.02	0.12	0.86
AIRPORT-TEMP-TIMEWINDOWS-CO	0.28	0.00	0.00	0.00	0.16	0.00	0.02	0.04	0.50	0.44
PIPESWORLD-NOTANKAGE-NONTEMP	0.16	0.10	0.00	0.00	0.74	0.00	0.00	0.00	0.00	1.00
PIPESWORLD-NOTANKAGE-TEMP	0.72	0.28	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
PIPESWORLD-TANKAGE-NONTEMP	0.12	0.00	0.00	0.00	0.54	0.00	0.00	0.00	0.34	0.66
PIPESWORLD-TANKAGE-TEMP	0.52	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.34	0.66
PIPESWORLD-NOTANKAGE-TEMP-DEAD	0.00	0.00	0.00	0.00	0.00	0.00	0.27	0.00	0.73	0.00
PROMELA-OPTICAL-TELEGRAPH	0.19	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.71	0.29
PROMELA-OPTICAL-TELEGRAPH-DP	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.40
PROMELA-OPTICAL-TELEGRAPH-FLUENTS	0.06	0.00	0.00	0.00	0.13	0.00	0.00	0.00	0.81	0.19
PROMELA-PHILOSOPHERS	0.58	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.40	0.60
PROMELA-PHILOSOPHERS-DP	0.94	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	1.00
PROMELA-PHILOSOPHERS-FLUENTS	0.02	0.00	0.00	0.00	0.19	0.00	0.00	0.79	0.00	0.21
PSR-SMALL	0.24	0.00	0.00	0.00	0.70	0.00	0.00	0.00	0.06	0.94
PSR-MIDDLE	0.98	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
PSR-MIDDLE-CO	0.26	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.72	0.28
PSR-LARGE	0.16	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.78	0.22
SATELLITE-STRIPS	0.53	0.06	0.00	0.00	0.25	0.00	0.00	0.00	0.17	0.83
SATELLITE-TIME	0.39	0.44	0.00	0.00	0.00	0.00	0.00	0.00	0.17	0.83
SATELLITE-TIME-TIMEWINDOWS	0.58	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.33	0.67
SATELLITE-TIME-TIMEWINDOWS-CO	0.53	0.03	0.00	0.00	0.11	0.00	0.00	0.00	0.33	0.67
SATELLITE-NUMERIC	0.44	0.00	0.00	0.00	0.11	0.00	0.00	0.03	0.42	0.55
SATELLITE-COMPLEX	0.36	0.22	0.00	0.08	0.08	0.03	0.00	0.06	0.17	0.77
SATELLITE-COMPLEX-TIMEWINDOWS	0.50	0.14	0.00	0.00	0.03	0.00	0.00	0.00	0.33	0.67
SATELLITE-COMPLEX-TIMEWINDOWS-CO	0.56	0.03	0.00	0.00	0.08	0.00	0.00	0.00	0.33	0.67
SETTLERS	0.10	0.00	0.00	0.00	0.85	0.00	0.00	0.00	0.05	0.95
UMTS-TEMP	0.96	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
UMTS-TEMP-TIMEWINDOWS	0.88	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
UMTS-TEMP-TIMEWINDOWS-CO	0.76	0.00	0.00	0.00	0.24	0.00	0.00	0.00	0.00	1.00
UMTS-FLAW-TEMP	0.02	0.88	0.00	0.00	0.10	0.00	0.00	0.00	0.00	1.00
UMTS-FLAW-TEMP-TIMEWINDOWS	0.00	0.44	0.00	0.00	0.10	0.00	0.46	0.00	0.00	0.54
UMTS-FLAW-TEMP-TIMEWINDOWS-CO	0.54	0.00	0.00	0.00	0.00	0.00	0.46	0.00	0.00	0.54
DEPOTS-STRIPS	0.27	0.27	0.05	0.00	0.41	0.00	0.00	0.00	0.00	1.00

Continued ...

Table 3: (continued)

Domain Variant	Instances Solvable by Both (F_b)						All Instances			
	F_i	F_q	F_t	F_w	F_{wt}	F_{wq}	F_n	F_g	F_u	F_b
DEPOTS-SIMPLETIME	0.23	0.68	0.05	0.00	0.00	0.00	0.00	0.05	0.00	0.95
DEPOTS-TIME	0.27	0.59	0.05	0.05	0.00	0.00	0.00	0.05	0.00	0.95
DEPOTS-NUMERIC	0.18	0.27	0.05	0.00	0.41	0.00	0.00	0.09	0.00	0.91
DRIVERLOG-STRIPS	0.70	0.10	0.00	0.00	0.00	0.00	0.00	0.10	0.10	0.80
DRIVERLOG-SIMPLETIME	0.60	0.20	0.00	0.00	0.00	0.00	0.00	0.10	0.10	0.80
DRIVERLOG-TIME	0.45	0.35	0.00	0.00	0.00	0.00	0.00	0.05	0.15	0.80
DRIVERLOG-NUMERIC	0.60	0.15	0.00	0.00	0.05	0.00	0.00	0.05	0.15	0.80
DRIVERLOG-HARDNUMERIC	0.55	0.20	0.00	0.00	0.05	0.00	0.00	0.05	0.15	0.80
FREECELL-STRIPS	0.05	0.10	0.00	0.05	0.70	0.00	0.00	0.10	0.00	0.90
ROVERS-STRIPS	0.70	0.00	0.00	0.00	0.30	0.00	0.00	0.00	0.00	1.00
ROVERS-SIMPLETIME	0.55	0.45	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
ROVERS-NUMERIC	0.45	0.05	0.00	0.00	0.10	0.00	0.00	0.25	0.15	0.60
SATELLITE-SIMPLETIME	0.75	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.00	1.00
SATELLITE-HARDNUMERIC	0.50	0.00	0.00	0.00	0.20	0.00	0.00	0.00	0.30	0.70
ZENOTRAVEL-STRIPS	0.85	0.00	0.00	0.00	0.15	0.00	0.00	0.00	0.00	1.00
ZENOTRAVEL-SIMPLETIME	0.80	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
ZENOTRAVEL-TIME	0.45	0.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
ZENOTRAVEL-NUMERIC	0.65	0.00	0.00	0.00	0.35	0.00	0.00	0.00	0.00	1.00
BLOCKSWORLD	0.57	0.29	0.06	0.03	0.06	0.00	0.00	0.00	0.00	1.00
Comparison between SGPlan$_{4.1}$ and LPG-TD-SPEED										
AIRPORT-NONTEMP	0.16	0.14	0.00	0.10	0.48	0.00	0.00	0.02	0.10	0.88
AIRPORT-TEMP	0.14	0.20	0.02	0.28	0.22	0.00	0.02	0.02	0.10	0.86
AIRPORT-TEMP-TIMEWINDOWS	0.08	0.22	0.00	0.30	0.26	0.00	0.00	0.04	0.10	0.86
PIPESWORLD-NOTANKAGE-NONTEMP	0.30	0.06	0.26	0.16	0.02	0.04	0.16	0.00	0.00	0.84
PIPESWORLD-NOTANKAGE-TEMP	0.44	0.00	0.36	0.02	0.00	0.00	0.18	0.00	0.00	0.82
PIPESWORLD-TANKAGE-NONTEMP	0.22	0.12	0.10	0.00	0.02	0.00	0.20	0.08	0.26	0.46
PIPESWORLD-TANKAGE-TEMP	0.24	0.06	0.12	0.02	0.00	0.00	0.22	0.06	0.28	0.44
PIPESWORLD-NOTANKAGE-TEMP-DEAD	0.07	0.03	0.03	0.07	0.03	0.00	0.03	0.53	0.20	0.24
PROMELA-OPTICAL-TELEGRAPH	0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.29
PROMELA-OPTICAL-TELEGRAPH-DP	0.25	0.00	0.00	0.00	0.00	0.00	0.15	0.00	0.60	0.25
PROMELA-PHILOSOPHERS	0.19	0.00	0.00	0.00	0.00	0.00	0.42	0.00	0.40	0.18
PROMELA-PHILOSOPHERS-DP	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
PSR-SMALL	0.40	0.54	0.00	0.00	0.00	0.00	0.00	0.04	0.02	0.94
PSR-MIDDLE	0.14	0.64	0.00	0.06	0.16	0.00	0.00	0.00	0.00	1.00
SATELLITE-STRIPS	0.42	0.36	0.03	0.00	0.03	0.00	0.00	0.17	0.00	0.83
SATELLITE-TIME	0.19	0.33	0.17	0.08	0.03	0.03	0.00	0.06	0.11	0.83
SATELLITE-TIME-TIMEWINDOWS	0.47	0.00	0.00	0.00	0.00	0.00	0.19	0.06	0.28	0.47
SATELLITE-NUMERIC	0.11	0.00	0.33	0.06	0.00	0.00	0.06	0.08	0.36	0.50
SATELLITE-COMPLEX	0.36	0.17	0.17	0.08	0.00	0.00	0.00	0.06	0.17	0.77
SATELLITE-COMPLEX-TIMEWINDOWS	0.44	0.00	0.03	0.00	0.00	0.00	0.19	0.06	0.28	0.53
SETTLERS	0.10	0.00	0.55	0.00	0.00	0.00	0.30	0.00	0.05	0.65
UMTS-TEMP	0.82	0.00	0.18	0.00	0.00	0.00	0.00	0.00	0.00	1.00
UMTS-TEMP-TIMEWINDOWS	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
UMTS-FLAW-TEMP	0.00	0.48	0.00	0.12	0.40	0.00	0.00	0.00	0.00	1.00
UMTS-FLAW-TEMP-TIMEWINDOWS	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
DEPOTS-STRIPS	0.32	0.36	0.05	0.18	0.09	0.00	0.00	0.00	0.00	1.00
DEPOTS-SIMPLETIME	0.09	0.09	0.27	0.50	0.00	0.00	0.00	0.05	0.00	0.95
DEPOTS-TIME	0.09	0.09	0.09	0.68	0.00	0.00	0.00	0.05	0.00	0.95
DEPOTS-NUMERIC	0.32	0.27	0.05	0.27	0.00	0.00	0.00	0.05	0.05	0.90
DRIVERLOG-STRIPS	0.65	0.15	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.80

Continued . . .

Table 3: (continued)

Domain Variant	Instances Solvable by Both (F_b)						All Instances			
	F_i	F_q	F_t	F_w	F_{wt}	F_{wq}	F_n	F_g	F_u	F_b
DRIVERLOG-SIMPLETIME	0.65	0.10	0.05	0.00	0.00	0.00	0.00	0.20	0.00	0.80
DRIVERLOG-TIME	0.65	0.10	0.05	0.00	0.00	0.00	0.00	0.20	0.00	0.80
DRIVERLOG-NUMERIC	0.60	0.20	0.00	0.00	0.00	0.00	0.00	0.15	0.05	0.80
DRIVERLOG-HARDNUMERIC	0.45	0.25	0.10	0.00	0.00	0.00	0.00	0.20	0.00	0.80
FREECELL-STRIPS	0.50	0.00	0.35	0.00	0.00	0.00	0.05	0.10	0.00	0.85
ROVERS-STRIPS	0.70	0.25	0.05	0.00	0.00	0.00	0.00	0.00	0.00	1.00
ROVERS-SIMPLETIME	0.75	0.20	0.05	0.00	0.00	0.00	0.00	0.00	0.00	1.00
ROVERS-NUMERIC	0.50	0.00	0.00	0.10	0.00	0.00	0.00	0.40	0.00	0.60
SATELLITE-SIMPLETIME	0.00	0.00	0.70	0.25	0.00	0.05	0.00	0.00	0.00	1.00
SATELLITE-HARDNUMERIC	0.15	0.00	0.55	0.00	0.00	0.00	0.00	0.00	0.30	0.70
ZENOTRAVEL-STRIPS	0.80	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
ZENOTRAVEL-SIMPLETIME	0.60	0.15	0.15	0.05	0.00	0.05	0.00	0.00	0.00	1.00
ZENOTRAVEL-TIME	0.65	0.00	0.30	0.05	0.00	0.00	0.00	0.00	0.00	1.00
ZENOTRAVEL-NUMERIC	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
BLOCKSWORLD	0.46	0.46	0.03	0.03	0.00	0.03	0.00	0.00	0.00	1.00
Comparison between SGPlan$_{4.1}$ and Downward										
AIRPORT-NONTEMP	0.52	0.00	0.02	0.16	0.18	0.00	0.00	0.12	0.00	0.88
PIPESWORLD-NOTANKAGE-NONTEMP	0.14	0.02	0.20	0.02	0.00	0.00	0.00	0.02	0.00	0.98
PIPESWORLD-TANKAGE-NONTEMP	0.16	0.00	0.16	0.00	0.02	0.00	0.32	0.06	0.28	0.34
PROMELA-OPTICAL-TELEGRAPH	0.00	0.00	0.00	0.00	0.00	0.00	0.29	0.00	0.71	0.00
PROMELA-OPTICAL-TELEGRAPH-DP	0.35	0.00	0.00	0.00	0.00	0.00	0.04	0.38	0.23	0.35
PROMELA-PHILOSOPHERS	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00	0.40	0.00
PROMELA-PHILOSOPHERS-DP	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
PSR-SMALL	0.42	0.04	0.00	0.00	0.48	0.00	0.00	0.06	0.00	0.94
PSR-MIDDLE	0.32	0.38	0.02	0.06	0.22	0.00	0.00	0.00	0.00	1.00
PSR-LARGE	0.12	0.04	0.00	0.04	0.02	0.00	0.00	0.40	0.38	0.22
SATELLITE-STRIPS	0.69	0.08	0.03	0.03	0.00	0.00	0.00	0.17	0.00	0.83
Comparison between SGPlan$_{4.1}$ and YAHSP										
AIRPORT-NONTEMP	0.24	0.22	0.02	0.10	0.12	0.00	0.18	0.02	0.10	0.70
PIPESWORLD-NOTANKAGE-NONTEMP	0.14	0.52	0.00	0.28	0.00	0.06	0.00	0.00	0.00	1.00
PIPESWORLD-TANKAGE-NONTEMP	0.22	0.32	0.02	0.02	0.04	0.00	0.04	0.24	0.10	0.62
PROMELA-OPTICAL-TELEGRAPH	0.27	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.71	0.27
PROMELA-PHILOSOPHERS	0.13	0.00	0.00	0.00	0.48	0.00	0.00	0.00	0.40	0.60
PSR-SMALL	0.36	0.02	0.00	0.02	0.54	0.00	0.00	0.02	0.04	0.94
SATELLITE-STRIPS	0.25	0.53	0.00	0.00	0.03	0.03	0.00	0.17	0.00	0.83
DEPOTS-STRIPS	0.50	0.32	0.05	0.00	0.00	0.00	0.14	0.00	0.00	0.86
DRIVERLOG-STRIPS	0.50	0.30	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.80
FREECELL-STRIPS	0.10	0.80	0.00	0.00	0.00	0.00	0.00	0.05	0.05	0.90
ROVERS-STRIPS	0.35	0.25	0.00	0.00	0.00	0.00	0.40	0.00	0.00	0.60
ZENOTRAVEL-STRIPS	0.30	0.65	0.00	0.00	0.00	0.05	0.00	0.00	0.00	1.00
BLOCKSWORLD	0.46	0.31	0.09	0.06	0.03	0.06	0.00	0.00	0.00	1.00

Keys:	(t_n, q_n)	(run time, quality) of SGPlan$_{4.1}$
	(t_g, q_g)	(run time, quality) of the target planner compared
	F_b	Fraction solved by both SGPlan$_{4.1}$ and the target planner ($F_b = F_i + F_q + F_t + F_w + F_{wt} + F_{wq} = 1 - F_n - F_g - F_u$)
	F_i	Fraction that $t_n \leq t_g$ and $q_n \leq q_g$ (SGPlan$_{4.1}$ has better or the same run time and quality)
	F_q	Fraction that $t_n > t_g$ and $q_n < q_g$ (SGPlan$_{4.1}$ has worse run time but better quality)
	F_t	Fraction that $t_n < t_g$ and $q_n > q_g$ (SGPlan$_{4.1}$ has worse quality but better run time)
	F_w	Fraction that $t_n > t_g$ and $q_n > q_g$ (SGPlan$_{4.1}$ has worse run time and worse quality)
	F_{wt}	Fraction that $t_n > t_g$ and $q_n = q_g$ (SGPlan$_{4.1}$ has worse run time but the same quality)
	F_{wq}	Fraction that $t_n = t_g$ and $q_n > q_g$ (SGPlan$_{4.1}$ has worse quality but the same run time)
	F_n	Fraction solved by SGPlan$_{4.1}$ but not by the target planner
	F_g	Fraction solved by the target planner but not by SGPlan$_{4.1}$
	F_u	Fraction unsolved by both SGPlan$_{4.1}$ and the target planner

Figures 17-20 further plot the time-quality trade-offs when the run time (*resp.* quality) of the target planner is normalized with respect to the corresponding measure of SGPlan$_{4.1}$ for all instances solvable by both planners. In each graph, we also list six percentages computed by normalizing F_i, F_t, F_q, F_w, F_{wt}, and F_{wq} with respect to F_b (defined in Table 3) for all the domains evaluated.

In the Airport domain, SGPlan$_{4.1}$ improves over or has the same performance as SGPlan$_4$ in terms of run time and quality for a majority (69.9%) of the instances (Figure 17a). In the NONTEMP variant, the solution files (not shown) show that SGPlan$_{4.1}$ cannot solve six ($F_g + F_u = 0.12$ in Table 3) of the seven largest instances (number 44 to 50); whereas Downward, the leading planner for this variant, can solve all 50 instances. SGPlan$_{4.1}$ has difficulty with these instances because the partitioned subproblems are too large to be evaluated by the embedded Metric-FF planner. This is also the reason for SGPlan$_{4.1}$ to be worse than Downward and LPG in terms of run time on the larger instances. An obvious solution is to employ a more efficient basic planner when it becomes available. In fact, this is one of the strengths of our partition-and-resolve approach. Another solution is to partition the subproblems further and to reduce their complexity to an extent that they can be handled by our modified Metric-FF planner. The design of such partitioning methods is still open at this time.

In the Pipesworld domain, SGPlan$_{4.1}$ has significant improvements over SGPlan$_4$ in terms of makespan on the NOTANKAGE-TEMP and TANKAGE-TEMP variants (Figure 17b). These improvements are due to the minimization of the estimated makespan ($\widetilde{T}$) in (19). However, no improvements were found on the NOTANKAGE-NONTEMP and TANKAGE-NONTEMP variants because (19) does not have a term that corresponds to the number of actions for the non-temporal variants. With respect to other planners, SGPlan$_{4.1}$ can solve more instances in the NOTANKAGE-NONTEMP, NOTANKAGE-TEMP, and TANKAGE-TEMP variants ($F_n - F_g \geq 0$ for all the corresponding rows in Table 3), and has consistently the shortest solution time in the NOTANKAGE-TEMP and TANKAGE-TEMP variants. For the NOTANKAGE-NONTEMP and TANKAGE-NONTEMP variants, YAHSP, however, can solve the most number of instances and has the shortest solution time in most cases, although it tends to produce longer plans. Last, as is discussed in Section 4.2, SGPlan$_{4.1}$ is not competitive in the PIPESWORLD-NOTANKAGE-TEMP-DEADLINE variant because it can only solve eight of the 30 instances.

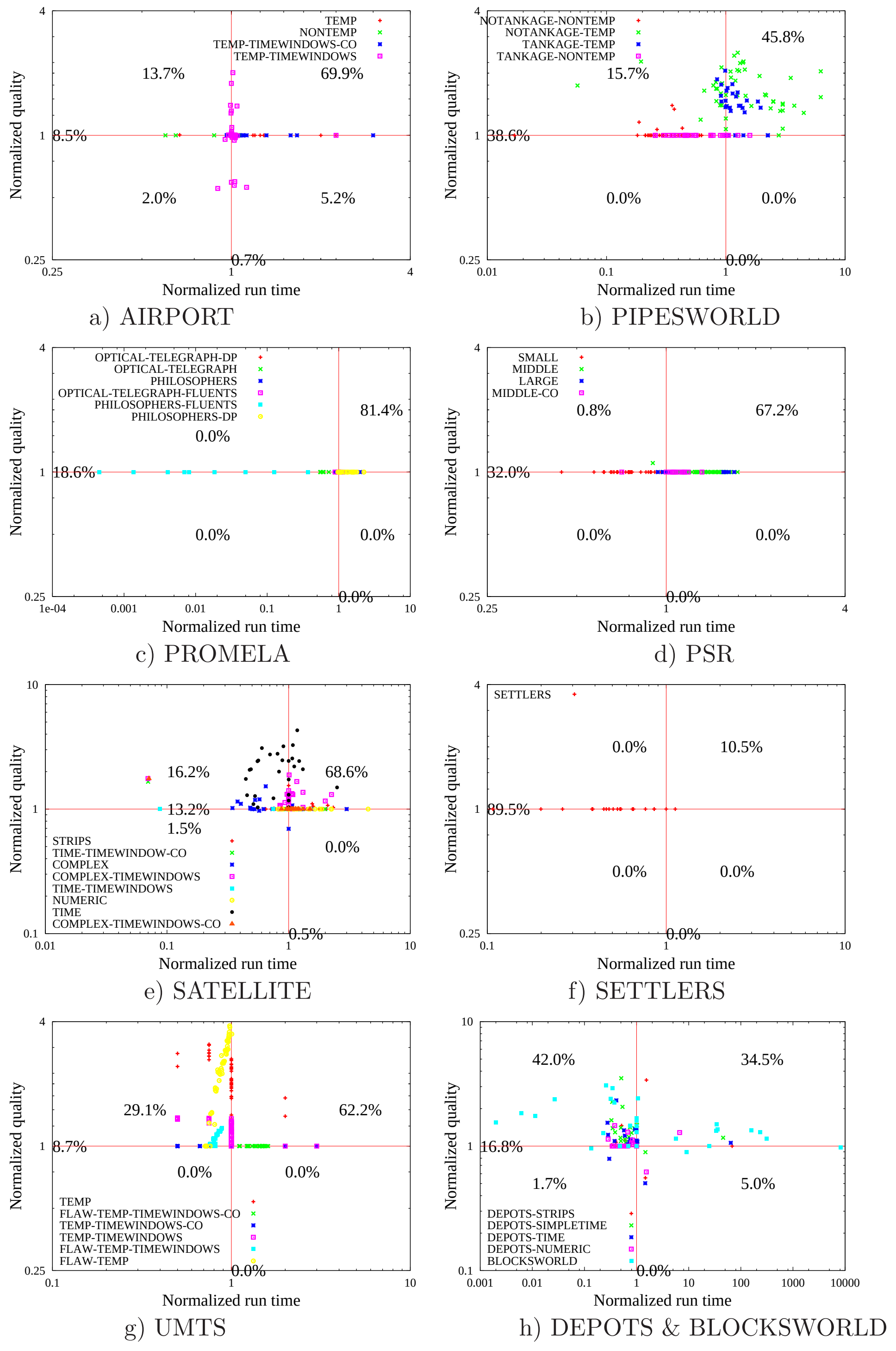


Figure 17: Run time-quality of SGPlan$_4$ on each instance normalized with respect to the corresponding run time-quality of SGPlan$_{4.1}$ on the same instance for all instances solvable by both planners. (Performance values larger than one are better for SGPlan$_{4.1}$.)

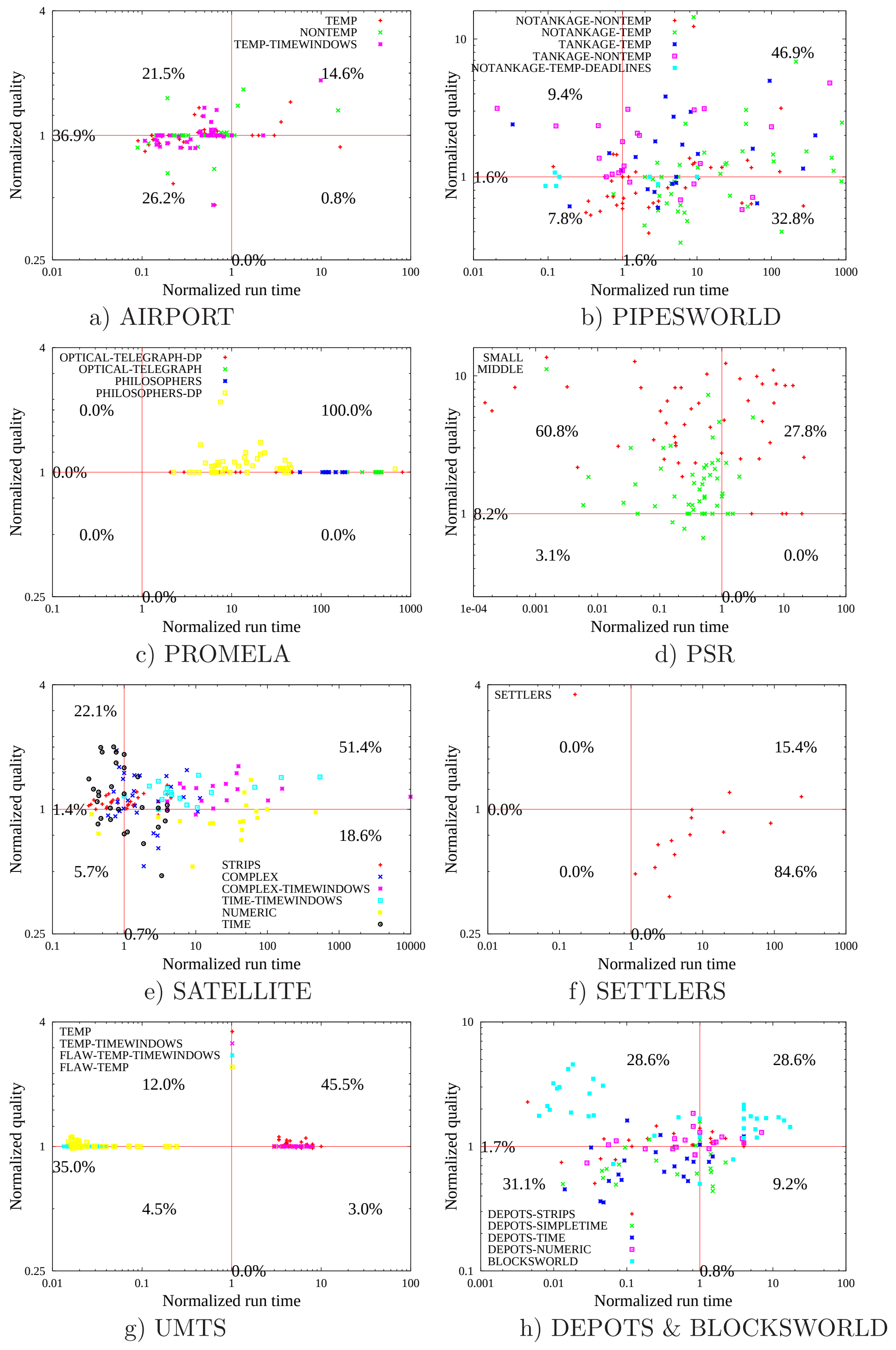


Figure 18: Run time-quality of LPG-TD-SPEED on each instance normalized with respect to the corresponding run time-quality of SGPlan$_{4.1}$ on the same instance for all instances solvable by both planners. (Performance values larger than one are better for SGPlan$_{4.1}$).

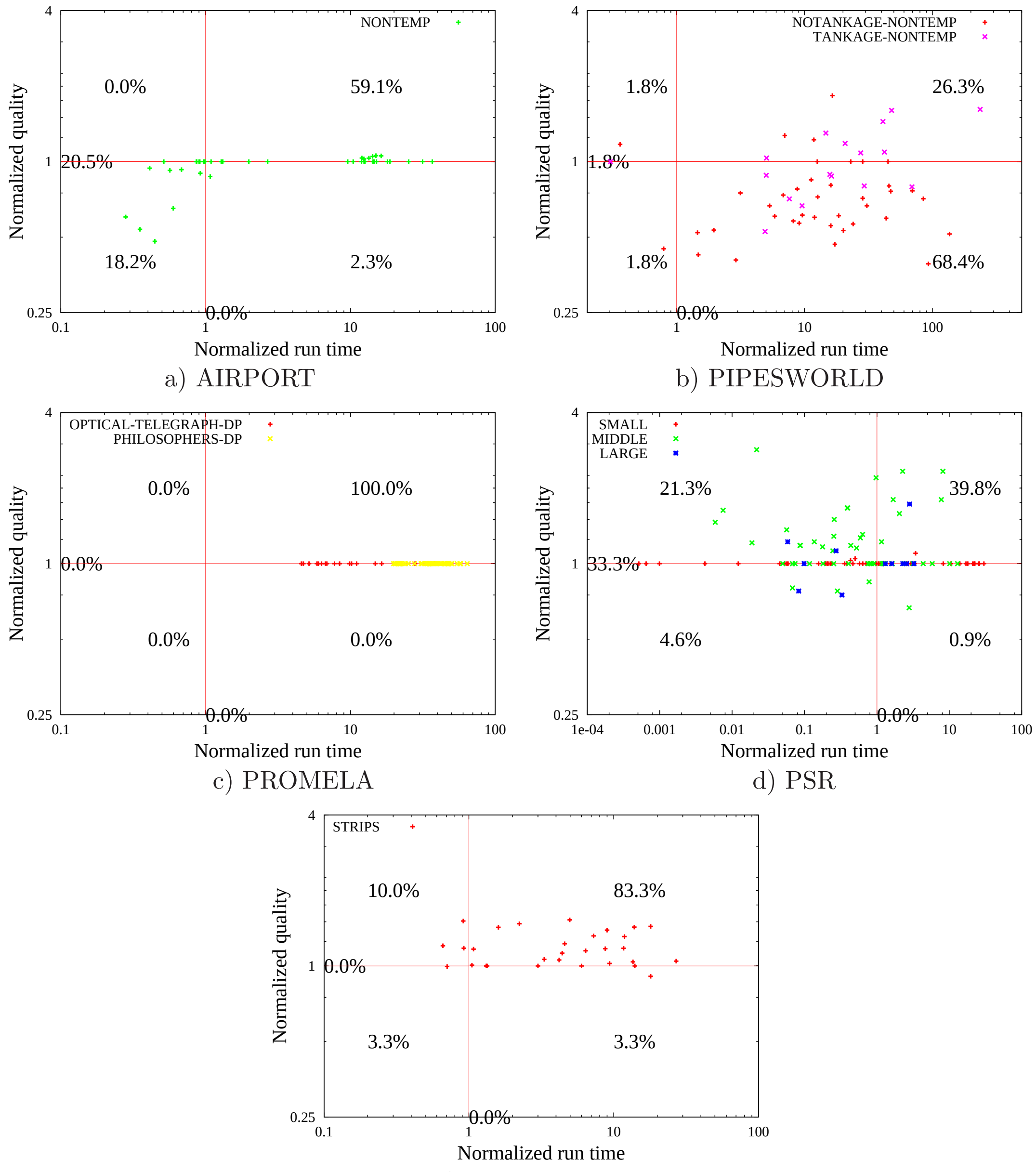


Figure 19: Run time-quality of Downward on each instance normalized with respect to the corresponding run time-quality of SGPlan$_{4.1}$ on the same instance for all instances solvable by both planners. (Performance values larger than one are better for SGPlan$_{4.1}$.)

In the Promela domain, SGPlan$_{4.1}$ has no improvements over SGPlan$_4$ in terms of quality but improves in terms of run time on instances that both can solve for four of the six variants (worse in the OPTICAL-TELEGRAPH-FLUENTS and PHILOSOPHERS-FLUENTS vari-

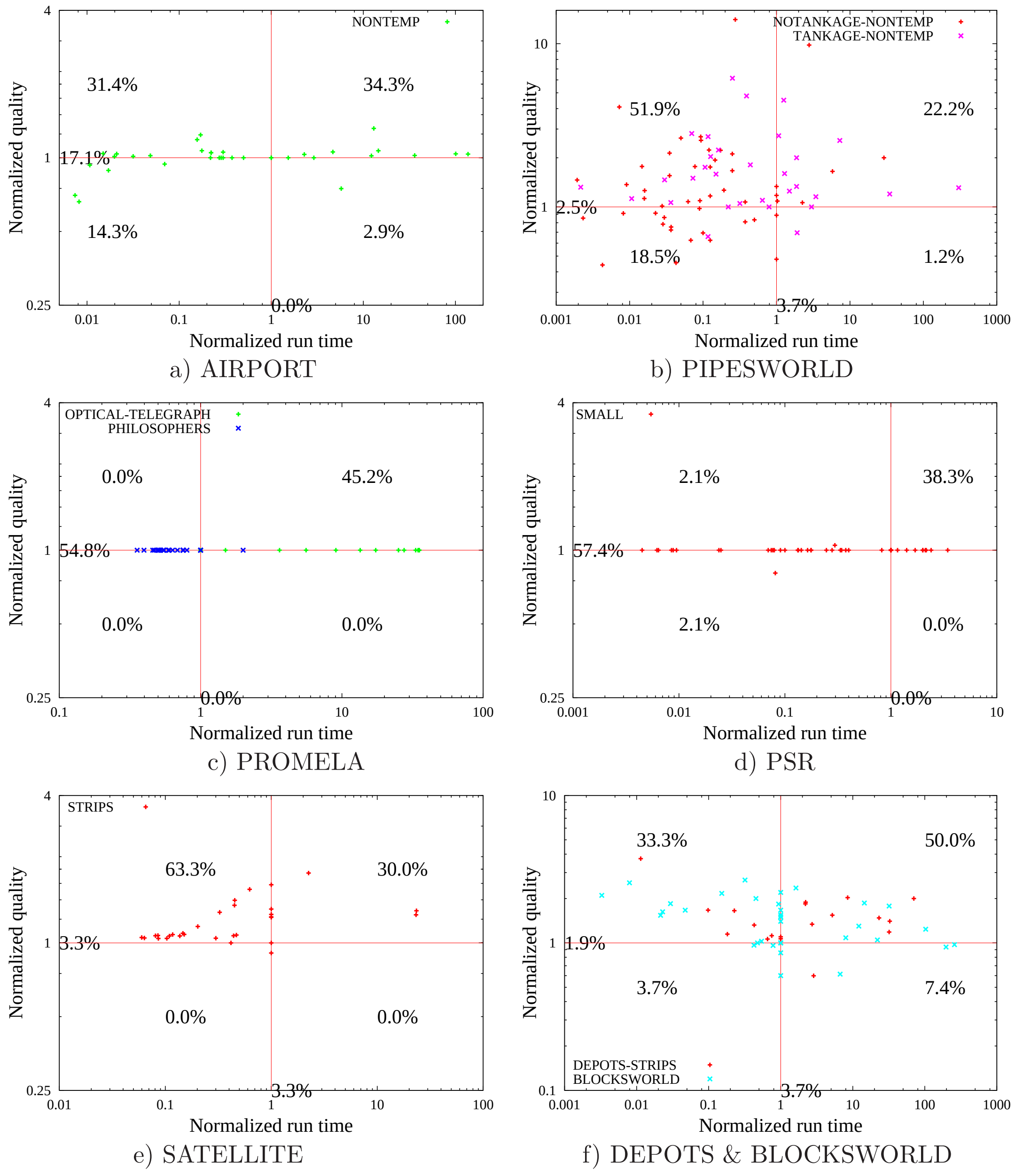


Figure 20: Run time-quality of YAHSP on each instance normalized with respect to the corresponding run time-quality of SGPlan$_{4.1}$ on the same instance for all instances solvable by both planners. (Performance values larger than one are better for SGPlan$_{4.1}$.)

ants). SGPlan$_{4.1}$ can solve the most number of instances in the OPTICAL-TELEGRAPH-FLUENTS, PHILOSOPHERS, PHILOSOPHERS-DP, and PHILOSOPHERS-FLUENTS variants when compared to LPG-TD-SPEED, Downward, and YAHSP. Further, it is the fastest planner in three of the variants but is slightly slower than YAHSP in the PHILOSOPHERS variant (Figures 18c, 19c, and 20c). In the OPTICAL-TELEGRAPH and OPTICAL-TELEGRAPH-DP variants, the organizer of IPC4 provided two versions, one written in pure STRIPS and another in ADL. However, there are only 14 (*resp.*, 19) instances in STRIPS and 48 (*resp.*, 48) instances in ADL for the OPTICAL-TELEGRAPH (resp., OPTICAL-TELEGRAPH-DP) variant. There are more instances available in ADL because ADL is space-efficient in its problem representation, whereas instances in STRIPS require large files. (For example, the file size of OPTICAL-TELEGRAPH-14 is 38 Kbytes in ADL and 8.3 Mbytes in STRIPS.) Since SGPlan$_{4.1}$ and SGPlan$_4$ cannot handle ADL at this time, they only solved those instances in pure STRIPS in these two variants. They were able to solve all the instances available in STRIPS and were the fastest in all these instances. However, Downward can handle instances in ADL and was able to solve more instances in these two variants. We plan to extend SGPlan$_{4.1}$ to directly support ADL in the future. Note that both SGPlan$_{4.1}$ and SGPlan$_4$ always find plans of the same or better quality for the instances solved in the OPTICAL-TELEGRAPH, OPTICAL-TELEGRAPH-DP, PHILOSOPHERS, and PHILOSOPHERS-DP variants when compared to the other three planners (Edelkamp & Hoffmann, 2004).

SGPlan$_{4.1}$ is the only planner that can solve some instances of all four variants of the PSR domain. Since PSR is a pure propositional domain, SGPlan$_{4.1}$ is unable to improve the solution quality over SGPlan$_4$. Nevertheless, the quality of SGPlan$_{4.1}$ is consistently better than all the other three planners ($F_i + F_q + F_{wt} > F_t + F_w + F_{wq}$ for all the corresponding rows in Table 3). In the SMALL variant, SGPlan$_{4.1}$ and LPG have comparable run times and cannot solve the few largest instances. Like the AIRPORT domain, SGPlan$_{4.1}$ has difficulty with the few largest instances because its basic planner cannot handle the partitioned subproblems. In the MIDDLE variant, SGPlan$_{4.1}$, LPG, and Downward can solve all 50 instances. The situation in the MIDDLE-CO and LARGE variants are similar to that in the OPTICAL-TELEGRAPH and the OPTICAL-TELEGRAPH-DP variants of the Promela domain. In these variants, Downward can handle directly the ADL format, but SGPlan$_{4.1}$ must expand the ADL syntax to pure STRIPS and exhausted its memory when evaluating the larger instances. We plan to address this issue in the future.

In the Satellite domain, SGPlan$_{4.1}$ has significant improvements in quality over SGPlan$_4$. In fact, SGPlan$_{4.1}$ generates solutions of better quality than all the other planners for most instances and can solve the most number of instances in seven variants. In the eighth variant (TIME), it was not able to solve the few largest instances because its memory usage exceeded 1 Gbytes. In all the variants except STRIPS, SGPlan$_{4.1}$ is faster than the other three planners. In the STRIPS variant, YAHSP is the fastest because it can generate multiple actions instead of a single action in each search step. However, it finds slightly longer plans when compared to those of SGPlan$_{4.1}$.

In the Settlers domain, SGPlan$_{4.1}$ does not improve the solution quality over SGPlan$_4$ because, as discussed earlier, (19) does not have a term that corresponds to the number of actions for non-temporal variants. SGPlan$_{4.1}$ can solve all the instances except the eighth instance, which we learned from the IPC4 organizers that it is an infeasible instance. It is also the fastest among all the planners, but generates longer plans than those of LPG-TD-SPEED. This is due to its iterative scheme for reducing producible resources. Because

Table 4: Summary on number of instances solved by the five planners compared ('?' means that it is not clear whether the domain can be solved because the object code was not available for testing, and '–' means that the planner does not support the language features in the benchmark.)

Domain		$\text{SGPlan}_{4.1}$	SGPlan_4	LPG-TD-SPEED	Downward	YAHSP
IPC4	Airport	154	156	134	50	36
	Pipesworld	174	166	158	60	93
	Promela	129	167	83	83	42
	PSR	122	122	99	131	48
	Satellite	204	207	157	36	36
	Settlers	19	19	13	–	–
	UMTS	300	254	200	–	–
	Total	1102	1091	844	360	219
IPC3	Depots	84	88	87	?	19
	DriverLog	80	87	99	?	20
	FreeCell	18	20	19	?	19
	Rovers	52	57	80	?	12
	Satellite	34	34	34	–	–
	ZenoTravel	80	80	80	?	20
	Total	348	366	399	?	90
IPC2	Blocksworld	35	35	35	?	35
Overall		**1485**	**1492**	**1243**	**360**	**344**

the optimal amount of resources cannot be found ahead of time, $\text{SGPlan}_{4.1}$ may incur some redundant actions for producing unused resources.

In the UMTS domain, $\text{SGPlan}_{4.1}$ can solve all the instances in all the six variants and is the fastest in four of them. Moreover, its makespans are greatly improved over those of SGPlan_4 by incorporating $\widetilde{T}$ in the modified heuristic function of Metric-FF, although its improvements in makespan over LPG-TD-SPEED are small for all the variants. $\text{SGPlan}_{4.1}$, however, is slower than LPG-TD-SPEED in the FLAW and FLAW-TIL variants. Its performance degradation in these variants is attributed to the flawed actions that can lead to overly optimistic heuristic values for relaxed-plan-based planners (Edelkamp & Hoffmann, 2004) like Metric-FF.

For the IPC3 Depots domain, $\text{SGPlan}_{4.1}$ has better quality than LPG-TD-SPEED and YAHSP in the STRIPS and NUMERIC variants, whereas the makespan of $\text{SGPlan}_{4.1}$ is worse than that of LPG-TD-SPEED for a majority of the instances in the TIME and SIMPLETIME variants. LPG-TD-SPEED is also faster than $\text{SGPlan}_{4.1}$ for a majority of the instances ($F_q + F_w + F_{wt} > F_i + F_t + F_{wq}$ for all the corresponding rows in Table 3). Due to the large fraction of initial active global constraints, the performance of subgoal partitioning in $\text{SGPlan}_{4.1}$ is unsatisfactory in this domain.

For the remaining IPC3 domains, $\text{SGPlan}_{4.1}$ generally improves SGPlan_4 in quality besides the Freecell domain which is in STRIPS. Except for the Satellite domain where LPG-TD-SPEED performs better, $\text{SGPlan}_{4.1}$ generates solutions with better quality for most of the instances. Further, SGPlan$_{4.1}$ is faster than LPG-TD-SPEED for more than half of the instances, although the difference in run times among the planners on these relatively easy instances is usually insignificant.

In the Blocksworld domain, SGPlan$_{4.1}$ generally finds solutions with a smaller number of actions than those of SGPlan$_4$, LPG-TD-SPEED, and YAHSP. However, SGPlan$_{4.1}$ is much slower than LPG-TD-SPEED on many instances because it needs more time for resolving the large fraction of initial active global constraints (Figure 18h).

8. Conclusions and Future Work

We have presented in this paper the partition-and-resolve approach and its application in SGPlan$_4$, a planner that won the first prize in the Suboptimal Temporal Metric Track and the second prize in the Suboptimal Propositional Track in IPC4. Table 4 summarizes the number of instances solved by the top planners in IPC4 as well as SGPlan$_{4.1}$. The results show that constraint partitioning employed by our planners is effective for solving a majority of the problems in the two competitions.

Our approach is based on the observation that the fraction of active mutex constraints across subgoals for a majority of the instances in IPC3 and IPC4 is very small. This observation allows us to partition the search into largely independent subproblems and to limit the amount of backtracking when resolving those violated global constraints across subproblems. The improvements are also attributed to a combination of techniques introduced for reducing the search space and for handling the new features in PDDL2.2.

In the future, we plan to study other partitioning techniques that can better exploit the constraint structure of planning domains. In particular, we will study fine-grain partitioning in order to address cases with a larger fraction of global constraints, and develop search strategies for solving problems with difficult-to-satisfy global constraints and deadlines. We also plan to extend our method to planning under uncertainty and to support more expressive modeling language features.

Acknowledgments

The research in this paper is supported by National Science Foundation Grant IIS 03-12084.

References

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*, 281–300.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence, Special issue on Heuristic Search*, *129*(1).

Chen, Y., & Wah, B. W. (2003). Automated planning and scheduling using calculus of variations in discrete space. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, pp. 2–11.

Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., & Tran, D. (2000). ASPEN - Automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps*. Space Operations Organization.

Doherty, P., & Kvarnstrm, J. (1999). Talplanner: An empirical investigation of a temporal logic-based forward chaining planner.. In *Proc. Sixth Int'l Workshop on Temopral Logic-based Forword Chaining Planner*, pp. 47–54. AIPS.

Edelkamp, S. (2002). Mixed propositional and numerical planning in the model checking integrated planning system. In *Proc. Workshop on Planning for Temporal Domains*. AIPS.

Edelkamp, S. (2003). Pddl2.2 planning in the model checking integrated environment. In *UK Planning and Scheduling Special Interest Group (PlanSig)*. Glasgow.

Edelkamp, S., & Hoffmann, J. (2004). Classical part, 4th international planning competition. `http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/`.

Foulser, D. E., Li, M., & Yang, Q. (1992). Theory and algorithms for plan merging.. *Artificial Intelligence*, *57*(2-3), 143–181.

Fourman, M. P. (2000). Propositional planning. In *Proc. Workshop on Model Theoretic Approaches to Planning*. AIPS.

Garrido, A., Fox, M., & Long, D. (2002). A temporal planning system for durative actions of pddl2.1. In *Proc. of European Conf. on Artificial Intelligence*, pp. 586–590.

Gerevini, A., & Serina, I. (2002). LPG: a planner based on local search for planning graphs with action costs. In *Proc. of the Sixth Int. Conf. on AI Planning and Scheduling*, pp. 12–22. Morgan Kaufman.

Hanks, S., & Weld, D. S. (1995). A domain-independent algorithm for plan adaptation.. *J. of Artificial Intelligence Research*, *2*, 319–360.

Helmert, M., & Richter, S. (2004). Fast downward - making use of causal dependencies in the problem representation. In *Proc. IPC4, ICAPS*, pp. 41–43.

Hoffmann, J. (2003). The metric-ff planning system: Translating ignoring delete lists to numeric state variables. *Journal of Artificial Intelligence Research*, *20*, 291–341.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, *14*, 253–302.

Jonsson, A. K., Morris, P. H., Muscettola, N., & Rajan, K. (2000). Planning in interplanetary space: Theory and practice. In *Proc. 2nd Int'l NASA Workshop on Planning and Scheduling for Space*. NASA.

Kambhampati, S., & Hendler, J. A. (1992). A validation-structure-based theory of plan modification and reuse.. *Artificial Intelligence*, *55*(2), 193–258.

Kautz, H., & Selman, B. (1996). Pushing the envelope: planning, propositional logic, and stochastic search. In *Proc. 13th National Conference on Artificial Intelligence*, pp. 1194–1201. AAAI.

Kautz, H., & Selman, B. (1999). Unifying SAT-based and graph-based planning. In *Proc. Int'l Joint Conf. on Artificial Intelligence*. IJCAI.

Kautz, H., & Walser, J. P. (2000). Integer optimization models of AI planning problems. *The Knowledge Engineering Review*, *15*(1), 101–117.

Koehler, J., & Hoffmann, J. (2000). On reasonable and forced goal ordering and their use in an agenda-driven planning algorithm. *J. of AI Research*, *12*, 339–386.

Lin, F. (2001). A planner called R. *AI Magazine*, 73–76.

Long, D., & Fox, M. (1998). Efficient implementation of the plan graph in STAN. *J. of AI Research*.

Nau, D., Muoz-Avila, H., Cao, Y., Lotem, A., & Mitchell, S. (2001). Total-order planning with partially ordered subtasks. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 425–430. IJCAI.

Nebel, B., Dimopoulos, Y., & Koehler, J. (1997). Ignoring irrelevant facts and operators in plan generation. In *Proc. European Conf. on Planning*, pp. 338–350.

Nebel, B., & Koehler, J. (1995). Plan reuse versus plan generation: A theoretical and empirical analysis.. *Artificial Intelligence*, *76*(1-2), 427–454.

Nigenda, R. S., Nguyen, X., & Kambhampati, S. (2000). AltAlt: Combining the advantages of Graphplan and heuristic state search. Tech. rep., Arizona State University.

Penberethy, J., & Weld, D. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pp. 103–114. KR Inc.

Penberethy, J., & Weld, D. (1994). Temporal planning with continuous change. In *Proc. 12th National Conf. on AI*, pp. 1010–1015. AAAI.

Porteous, J., Sebastia, L., & Hoffmann, J. (2001). On the extraction, ordering, and usage of landmarks in planning. In *Proc. European Conf. on Planning*, pp. 37–48.

Refanidis, I., & Vlahavas, I. (2001). The GRT planner. *AI Magazine*, 63–66.

Refanidis, I., & Vlahavas, I. (2002). The MO-GRT system: Heuristic planning with multiple criteria. In *Proc. Workshop on Planning and Scheduling with Multiple Criteria*. AIPS.

Subbarao, M. B. D., & Kambhampati, S. (2002). Sapa: A domain-independent heuristic metric temporal planner. Tech. rep., Arizona State University.

Tate, A., Drabble, B., & Kirby, R. (1994). O-Plan2: an open architecture for command, planning and control. *Intelligent Scheduling*, 213–239.

Tsamardinos, I., Pollack, M. E., & Horty, J. F. (2000). Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches.. In *Proc. Int'l Conf. on AI Planning and Scheduling (AIPS)*, pp. 264–272.

Wah, B., & Chen, Y. (2006). Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*, *170*(3), 187–231.

Wah, B. W., & Chen, Y. (2003). Partitioning of temporal planning problems in mixed space using the theory of extended saddle points. In *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence*, pp. 266–273.

Wah, B. W., & Chen, Y. (2004). Subgoal partitioning and global search for solving temporal planning problems in mixed space. *Int'l J. of Artificial Intelligence Tools*, *13*(4), 767–790.

Wilkins, D. (1990). Can AI planners solve practical problems?. *Computational Intelligence*, 232–246.

Wolfman, S., & Weld, D. (2000). Combining linear programming and satisfiability solving for resource planning. *The Knowledge Engineering Review*, *15*(1).

Yang, Q. (1997). *Intelligent planning: a decomposition and abstraction based approach*. Springer-Verlag, London, UK.