Preferring Properly: Increasing Coverage while Maintaining Quality in Anytime Temporal Planning

Patrick Eyerich[1]

Abstract. Temporal Fast Downward (TFD) is a successful temporal planning system that is capable of dealing with numerical values. Rather than decoupling action selection from scheduling, it searches directly in the space of time-stamped states, an approach that has shown to produce plans of high quality at the price of coverage. To increase coverage, TFD incorporates deferred evaluation and preferred operators, two search techniques that usually decrease the number of heuristic calculations by a large amount. However, the current definition of preferred operators offers only limited guidance in problems where heuristic estimates are weak or where subgoals require the execution of mutex operators. In this paper, we present novel methods for refinement of this definition and show how to combine the diverse strengths of different sets of preferred operators using a restarting procedure incorporated into a multi-queue best-first search. These techniques improve TFD's coverage drastically and preserve the average solution quality, leading to a system that solves more problems than each of the competitors of the temporal satisficing track of IPC 2011 and clearly outperforms all of them in terms of IPC score.

1 Introduction

Temporal Planning is an important generalization of classical planning that allows modeling of many applications more realistically by taking into account not only causal dependencies between actions but also their temporal interactions. It is a growing research area and there are many interesting approaches that tackle its challenges. LPG [11] is based on stochastic local search in the space of *action graphs*. Crikey3 [5] employs heuristic forward search interleaving planning and scheduling via Simple Temporal Networks. CPT4 [24] is a planning system based on partial order causal links that is optimal for the conservative semantics of Smith and Weld [23].

A common approach to temporal planning, as for example taken by SGPlan [16], YAHSP2 [25] and $\text{DAE}_{\text{YAHSP}}$ [9], is to consider only logical dependencies between actions first while temporal dependencies are taken into account just afterwards to find an appropriate scheduling for the chosen actions. While having the potential of being very fast due to the possibility of utilizing successful techniques from the much more investigated field of classical planning, such approaches are doomed to fail in temporally expressive domains [6], and suffer from severe drawbacks in temporally simple problems, as choosing the wrong actions might render the final solutions to be purely sequential and therefore of very low quality.

Another approach as for example taken by Sapa [8], LMTD [17], or Temporal Fast Downward (TFD) [10], is to perform forward search in the space of time-stamped states, where at each search state either a new action can be started or time can be advanced to the end point of an already running action, thereby combining action selection and scheduling. Also, POPF2 [2, 3] performs a forward search, using a partial order rather than a total order like Sapa and TFD do. While these approaches are usually very good in terms of quality, their coverage on current benchmarks is typically relatively low.

As a first step to increase its coverage, TFD incorporates preferred operators and deferred evaluation [21]. The general idea of preferred operators is to favor operators that are part of a solution for the heuristic abstraction of the problem. Deferred evaluation postpones heuristic computations from the point where a search node is generated to the point where it is expanded, rating nodes with the estimate of their predecessor during search. Thereby, the number of heuristic calculations is decreased by a large amount at the price of informativeness. Even more than in classical planning, in temporal planning the heuristic computation is the bottleneck of search, and indeed it turns out that the use of deferred evaluation and preferred operators increases the performance of TFD enormously. Unfortunately, this improvement does not occur uniformly over all planning domains. Instead, there are problems where using preferred operators and deferred evaluation worsens results.

The first contribution of this paper consists of novel methods that strengthen the selection criteria for preferred operators. Different methods have strengths on different domains and some of them clearly increase TFD's coverage on their own. However, all of the new methods have a common drawback: They produce solutions of lower quality than the original definition. This leads to the second contribution, a restarting procedure embedded in a multi-queue best first search that, besides further increasing coverage, regains the lost quality. Our resulting system is able to overcome the disadvantage of searching in the space of time stamped states, i.e., low coverage, while maintaining its major advantage, high solution quality.

The remainder of this paper is structured as follows: In the next section we describe TFD with an emphasis on its heuristic. The subsequent section presents narrowing strategies for preferred operators and a multi-queue search algorithm featuring restarts. Afterwards, we present detailed experiments before we conclude.

2 Temporal Fast Downward

Temporal Fast Downward (TFD) [10] is a domain-independent progression search planner built on top of the classical planner *Fast Downward* [12]. It extends the original system by supporting durative actions as well as numeric and object fluents. TFD solves a planning task in three phases: As a first step, the Boolean PDDL encoding is *translated* into a finite-domain representation similar to SAS^+ [1, 13]. Afterwards, in a *knowledge compilation* step, several internal data structures are generated. The scope of this paper is the third part, a best-first progression *search*. For the sake of simplicity,

[1] University of Freiburg, Germany, email: eyerich@informatik.uni-freiburg.de

we only deal with non-numerical fluents. Note, however, that all our results can be generalized to the numeric case straightforwardly.

In the following, we use the definition of Eyerich et. al. of a *temporal SAS$^+$ planning task* [10], a tuple $\Pi = \langle \mathcal{V}, s_0, s_\star, \mathcal{A}, \mathcal{O} \rangle$, where $\mathcal{V}$ is a set of *state variables*. The initial state s_0 is given by a variable assignment (a *state*) over all fluents in $\mathcal{V}$ and the set of goal states $s_\star$ is defined by a partial state (a state restricted to a subset of fluents) over $\mathcal{V}$. Analogously to the Boolean setting, we identify such variable mappings with the *set of atoms* $v{=}w$ that they make true. For an atom x we write $\mathrm{var}(x)$ to denote the variable associated with x. $\mathcal{A}$ is a finite set of *axioms* and $\mathcal{O}$ is a finite set of *durative actions*.

A *time-stamped state* $\mathcal{S} = \langle t, s, E, C_\leftrightarrow, C_\dashv \rangle$ consists of a *time stamp* $t \geq 0$, a *state* s, a set E of *scheduled effects*, and two sets $C_\leftrightarrow$ and $C_\dashv$ of persistent and end conditions.

A durative action is *applicable* in a time-stamped state $\mathcal{S}$ if it can be integrated into $\mathcal{S}$ in a consistent way [10]. The successors of a time-stamped state are generated by either inserting an applicable durative action at the current time point or by increasing the timestamp to the earliest time point where a scheduled action ends.

2.1 Context-enhanced additive heuristic

For guiding the search, TFD uses a variant of the (inadmissible) context-enhanced additive heuristic (h^{cea}) [14] extended to cope with numeric variables and durative actions. To make h^{cea} useful for temporal planning, Eyerich et. al. show how to transform durative actions to several types of so-called *instant actions* [14], which we assume to be given in this paper. Instant actions are sets of effects of the form $v{=}w', z \to v{=}w$, where v is a variable, z is a partial state not mentioning v, and w and w' are values for v. Such an effect means that if the current state s satisfies z and maps v to w', then the successor state s', resulting from the application of the operator, maps v to w (while all mappings that are not changed by any effect of the operator stay the same). We also write $a : v{=}w', z \to v{=}w$ to make clear that the rule is an effect of the instant action a.

Given a state s and an atom $v{=}w$, we denote with $s[v{=}w]$ the state that is like s except for variable v, which it maps to w. Similarly, we write $s[s']$ where s' is a partial state to denote the state that is like s' for variables defined in s' and like s for all other variables.

For a time-stamped state s and a goal specification $s_\star$, the cost-sensitive variant of h^{cea} is defined as

$$h^{cea}(s) \overset{\text{def}}{=} \sum_{x \in s_\star} h^{cea}(x|x_s),$$

where x_s is the atom that refers to $\mathrm{var}(x)$ in state s and $h^{cea}(x|x_s)$ estimates the costs of changing the value of $\mathrm{var}(x)$ from the value it has in s to the one required in $s_\star$.

The context-enhanced additive heuristic makes the underlying assumption that for any atom x conditions referring to $\mathrm{var}(x)$ are achieved first, while all other conditions are evaluated in the resulting state s'', leading to the following definition:

$$h^{cea}(x|x') \overset{\text{def}}{=} \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o:x'',z \to x} \Big(c(o, s'') + h^{cea}(x''|x') + \sum\limits_{x_i \in z} h^{cea}(x_i|x_i'')\Big) & \text{else} \end{cases}$$

where $c(o, s)$ is the cost of applying operator o in state s. The state s'' is the state after reaching x'' from x'. Note that with the minimum of the empty set being infinity, $h^{cea}(x|x')$ might also be infinity and if it is, there is no plan that satisfies the goal in the original task.

In this definition, the first case is trivial. In the second case, the first summand, $c(o, s'')$, captures the cost of applying the minimizing operator o in state s'', the second one estimates the cost of achieving x'' from x', and the third one the cost of making all other conditions z of the rule true. In this third term, atom x_i'' is the atom associated with $\mathrm{var}(x_i)$ in the state that results from achieving x'' from x'.

To reschedule solutions in order to reduce their makespan, the TFD version used for this paper features a partial-order lifting procedure that is inspired by the work of Do and Kambhampati and Coles et. al. [7, 4] and extended to be able to deal with conditional effects.

3 Preferred Operators

Conceptually, the idea of preferred operators is to transfer information about which operator's application seems to be promising from the heuristic abstraction to the actual search. This concept was first realized by McDermott by determining *favored actions* in the context of greedy regression graphs as those applicable actions that are part of the minimal cost subgraph achieving the goals [19, 20]. Hoffmann and Nebel defined *helpful actions* in their FF planner as those actions that achieve a fact required by an action in the relaxed plan that appears in the first layer of the planning graph [15]. FF considers only helpful actions in its first attempt of finding a solution and switches to a complete greedy best-first search if it fails. Another approach is used in Fast Downward, where besides the usual open list containing all applicable operators there is a separate open list containing only preferred operators. Different strategies of how to best combine these two open list have been investigated [21, 22].

Using the definition of the context-enhanced additive heuristic, the set $\mathcal{P}(s)$ of *preferred operators* is defined as

$$\mathcal{P}(s) \overset{\text{def}}{=} \bigcup_{x \in s_\star} \mathcal{P}(x|x_s),$$

where

$$\mathcal{P}(x|x') \overset{\text{def}}{=} \begin{cases} \{\} & \text{if } x = x' \text{ or } h^{cea}(x|x') = \infty \\ \{o\} & \text{if } \exists\, o : x', w \to x : \\ & \quad h^{cea}(x|x') = c(o, s') \\ \bigcup\limits_{x_i \in w} \mathcal{P}(x_i|x_i') & \text{if } \exists\, o : x', w \to x : \\ & \quad h^{cea}(x|x') = \Big(c(o, s') + \sum\limits_{x_i \in w} h^{cea}(x_i|x_i')\Big) \\ \mathcal{P}(x''|x') & \text{if } \exists\, x'' : h^{cea}(x''|x') \\ & \quad + h^{cea}(x|x'') = h^{cea}(x|x') \end{cases}$$

Each condition additionally requires the previous conditions to be unsatisfied. We furthermore assume that no action with zero cost exists and that, if the existentially quantified conditions are satisfied for more than one operator or atom, an arbitrary one is chosen according to a fixed tie-breaking strategy.

The first case is trivial. The second case defines an operator o that transforms x' to x with cost equal to $h^{cea}(x|x')$ (which means that all its preconditions have to be satisfied) as preferred. The third case is similar to the second one except that some of the operator's preconditions are not satisfied. In that case, preferred operators are recursively defined over those preconditions. In the last case, x' cannot be changed to x by a single operator but only via an intermediate state, so preferred operators are recursively defined over this state.

In its default configuration, TFD uses a straight-forward adaptation of the boosted dual queue approach for preferred operators of Fast Downward [12]. As can be seen in the experiments section, preferred operators work best in the context of deferred evaluation. However, there are certain domain characteristics for which that is not the case. Especially in problems where goals are conflicting, requiring mutex operators, the preferred operator handling of TFD does not yield good results in the context of deferred evaluation.

The main reason for this poor behavior is that h^{cea} computes costs of subgoals independently from each other. In that way, a set of preferred operators might contain mutex operators each leading to a successor state with the same heuristic estimate (due to deferred evaluation) while the successors of each successor have a higher heuristic estimate. To see this, think of a problem in an elevators domain where we have two goals g_1 and g_2 to transport two passengers p_1 and p_2 from their common starting location f_5 to their desired floors f_1 and f_{10}, respectively. When investigating the subproblems independently from each other, as h^{cea} does, it might be meaningful to use the same elevator e_1, located at f_5, to transport both p_1 and p_2. In such a situation, both the operators move-down(e_1, f_5, f_1), leading to state s_1, and move-up(e_1, f_5, f_{10}), leading to state s_2, are preferred, and since we use deferred evaluation, s_1 and s_2 share the same heuristic estimation. When s_1 is expanded, however, e_1 has started to move to f_1 in order to satisfy g_1, and the heuristic realizes that g_2 becomes more expensive (potentially to a higher degree than the amount that g_1 becomes cheaper), leading to a worse overall state evaluation for all successors of s_1. Expansion of s_2 is analogous. In such a situation a potentially very large set of states has to be visited before the search actually progresses in the right direction.

3.1 Narrowing strategies

Our new selection strategies are basically methods to intelligently narrow the set of preferred operators, motivated by examples like the one above: If by using only preferred operators a planning task is rendered incomplete anyway, and if generating preferred operators for all subgoals at once can lead to situations where the search gets stuck, why not limit ourselves to generating preferred operators for only up to n subgoals? Of course, the obvious questions are *which* and *how many* operators out of a set of preferred ones we should choose. We have found three narrowing strategies to be useful in practice: To utilize only the preferred operators that correspond to the first n yet unsatisfied goals, called $\mathcal{O}$, or to choose the preferred operators corresponding to the n goals that are cheapest or most expensive to satisfy according to the heuristic, called $\mathcal{C}$ and $\mathcal{E}$, respectively. More concretely, a *narrowing strategy* $\mathcal{X}^n(s)$ is defined as

$$\mathcal{X}^n(s) \overset{\text{def}}{=} \bigcup_{x \in X \subseteq s_\star} \mathcal{P}(x|x_s)$$

with an appropriate X of cardinality n chosen according to the selection strategy of $\mathcal{X}$. For $\mathcal{O}$, this strategy is defined such that $x \leq_{\mathcal{O}} y$ for all $x \in X, y \in (s_\star \setminus X)$ holds for some appropriate ordering relation $\leq_{\mathcal{O}}$. For $\mathcal{C}$ it has to hold that $h^{cea}(x|x_s) \leq h^{cea}(y|y_s)$ for all $x \in X, y \in (s_\star \setminus X)$, and for $\mathcal{E}$ it has hold that $h^{cea}(x|x_s) \geq h^{cea}(y|y_s)$ for all $x \in X, y \in (s_\star \setminus X)$.

Basically, all narrowing strategies examine the current state s and choose up to n goals x_i from $s_\star$ to compute preferred operators for: $\mathcal{O}$ determines the first n unsatisfied goals (according to an appropriate ordering relation $\leq_{\mathcal{O}}$), while $\mathcal{C}$ and $\mathcal{E}$ determine the heuristic cost of each subgoal as if it would be the only goal to satisfy (as said, this is done by h^{cea} anyway) and choose the n that are cheapest and most expensive, respectively. Note that with small n the search is driven to satisfy the goals more sequentially, however, each goal might be satisfied by parallel action applications.

Finding a good ordering relation for $\mathcal{O}$ is very much related to the more general task of detecting goal orderings [18]. In this paper, we only use the natural ordering that is defined by the order in which variables occur in the problem description for that purpose and defer the interesting question of how to combine our technique with goal ordering detection techniques to future work.

3.2 Priority based multi-queue search with restarts

As we will show in the experiments section, utilizing our new techniques in TFD pays off in terms of coverage. Unfortunately, the produced solutions are typically of a lower quality than those of the original definition as the search is driven to satisfy goals more sequentially. Additionally, it can be observed that the different narrowing strategies have strengths in different domains. Motivated from these two facts, we have developed an algorithm that incorporates several narrowing strategies into a best-first search framework that uses an own open list for each strategy, as outlined in Algorithm 1.

Algorithm 1: Best-first search with restarts, deferred evaluation, and several open lists in a priority based multi-queue approach.

```
1  activeList = chooseOpenListToStartWith()
2  forall open in openLists do
3      open.priority = 0
4  activeList.priority += $V$
5  activeList.add($s_0$)
6  closedList ← $\emptyset$
7  lastProgressAtStep = 0, currentStep = 0
8  while activeList is not empty do
9      currentStep += 1
10     if (currentStep - lastProgressAtStep) $> K$ then
11         activeList = nextOpenListToBoost()
12         restartAtLine2() or switchToRoundRobinMode()
13     $s$ ← activeList.pop()
14     activeList.priority -= 1
15     if $s \notin$ closedList then
16         closedList.add($s$)
17         if $s \models G$ then
18             return $s$ as solution
19         h = $s$.compute_heuristic()
20         f = $s$.timestamp + $h$
21         if makes_progress(s) then
22             activeList.priority += $V$
23             lastProgressAtStep = currentStep
24         forall child states $s'$ of $s$ do
25             forall open in openLists do
26                 if open.matches($s'$) then
27                     open.add($s'$, $f$)
28     activeList = selectList()
29 return no solution found
```

The algorithm is based on the boosted dual-queue best-first search approach of Fast Downward [12]. It maintains a set of open lists, each associated with a corresponding selection method. It has been shown that alternating between different open lists is a good idea if the open lists contain operators ordered by different heuristics [22]. In our context, however, alternating did not work well, so we have chosen a priority based approach where each open list is associated with a priority and at each search step the algorithm selects the non-empty list with the highest priority (line 28). The search keeps track of the number of steps that were performed since the last time progress has been made (progress is made if a state is extracted from a list that has a lower heuristic estimate than each other state that has previously been taken from that list). If more than K steps have been made without making progress, the search restarts (lines 10–12), boosting a different open list each time by giving it a high initial priority while all other lists start with priority zero. If the search has restarted with each open list being boosted initially once, it switches to a round robin selection mode (line 12, details have been omitted from the pseudocode to ensure readability). During successor generation, nodes are inserted into the appropriate open lists according to their associated selection strategies (lines 24–27). Note that using a regular open list containing all applicable successors (which is done in our implementation) ensures completeness of the algorithm on temporally simple problems.

For the two parameters of the algorithm we have found $K = 3000$ and $V = 1000$ to work well in practice (these parameters, however, are quite robust and we got reasonable results for a wide range of values for both K and V). Note that the algorithm can be called from outside in an anytime fashion where the makespan of previously found solutions can be used to prune the search space.

4 Experiments

In our first experiment[2] we show the influence that deferred evaluation and preferred operators have on the search performance of TFD.

IPC 2011	e	d	$\mathcal{P}$e	$\mathcal{P}$d
CREWPLANNING	0.0 (0)	0.0 (0)	**19.9** (**20**)	**19.9** (**20**)
ELEVATORS	0.0 (0)	0.0 (0)	0.0 (0)	**1.0** (**1**)
FLOORTILE	4.0 (**5**)	4.1 (**5**)	**4.9** (**5**)	**4.9** (**5**)
MATCHCELLAR	1.0 (1)	1.0 (1)	**15.6** (**20**)	**15.6** (**20**)
OPENSTACKS	**17.8** (**20**)	16.7 (**20**)	**17.8** (**20**)	17.7 (**20**)
PARCPRINTER	0.0 (0)	0.0 (0)	0.0 (0)	0.0 (0)
PARKING	**13.7** (**17**)	10.2 (14)	7.1 (9)	12.2 (16)
PEGSOL	17.9 (**18**)	**18.0** (**18**)	17.9 (**18**)	17.9 (**18**)
SOKOBAN	**2.9** (**3**)	**2.9** (**3**)	**2.9** (**3**)	**2.9** (**3**)
STORAGE	0.0 (0)	0.0 (0)	0.0 (0)	0.0 (0)
TMS	0.0 (0)	0.0 (0)	0.0 (0)	0.0 (0)
TURNANDOPEN	12.0 (14)	10.7 (13)	12.5 (17)	**13.3** (**20**)
Overall	69.2 (78)	63.7 (74)	98.6 (112)	**105.3** (**123**)

Table 1. Results on IPC 2011 benchmarks measuring the influence of deferred evaluation and preferred operators in regular TFD, showing IPC score and coverage (in parentheses). Used abbreviations are '$\mathcal{P}$' for preferred operators, 'd' for deferred evaluation, and 'e' for eager evaluation.

Results showing IPC score[3] and coverage (in parentheses) on IPC 2011 benchmarks are presented in Table 1. Without preferred operators, switching from eager ('e') to deferred evaluation ('d') speeds up the search by saving a lot of heuristic computations but reduces guidance, altogether leading to a slightly worse performance. This drawback, however, can be overcome by incorporating preferred operators ('$\mathcal{P}$d'). With preferred operators bringing back some of the lost guidance, the advantages of the reduced computational effort of deferred evaluation can be fully exploited, leading to both a much higher coverage and IPC score. Note that using preferred operators increases performance also in the context of eager evaluation ('$\mathcal{P}$e'), but to a lesser degree, so combining preferred operators and deferred evaluation as in the original TFD setting clearly is the best option.

For our second experiment we have implemented the methods presented in Section 3 to narrow the set of preferred operators. Results for $\mathcal{C}^n$, $\mathcal{E}^n$, and $\mathcal{O}^n$ with $1 \leq n \leq 3$ are shown in Table 2.

[2] All our experiments were run on AMD Opteron 2.3 GHZ Dual Core processors with a memory limit of 2 GB and a timeout of 30 minutes.

[3] If Q^* is the makespan of a reference solution, a planner producing a solution of makespan Q receives Q^*/Q points of IPC score. For all our experiments the best known plans (including ours) are used as reference.

IPC 2011	$\mathcal{C}^1$	$\mathcal{C}^2$	$\mathcal{C}^3$	$\mathcal{E}^1$	$\mathcal{E}^2$	$\mathcal{E}^3$	$\mathcal{O}^1$	$\mathcal{O}^2$	$\mathcal{O}^3$
CREWPLANNING	14.2	15.5	**15.6**	0.0	2.4	1.6	0.0	2.4	4.2
	19	**20**	**20**	0	3	2	0	3	5
ELEVATORS	**15.1**	10.5	7.5	0.0	0.0	0.0	13.4	7.0	4.5
	18	12	8	0	0	0	**18**	10	6
FLOORTILE	4.3	4.9	4.7	4.4	**5.2**	4.5	4.8	4.8	4.0
	5	5	5	5	**6**	5	5	5	4
MATCHCELLAR	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**
	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
OPENSTACKS	3.6	5.0	6.4	14.2	14.4	**15.3**	4.0	6.1	7.8
	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
PARCPRINTER	1.0	0.0	0.0	0.0	0.0	0.0	**9.4**	2.7	1.7
	1	0	0	0	0	0	**10**	3	2
PARKING	14.0	**14.9**	11.4	8.9	8.6	9.0	6.7	8.5	7.5
	17	**19**	14	12	12	12	9	11	10
PEGSOL	17.7	17.4	18.5	18.6	18.8	18.8	18.4	19.0	**19.3**
	18	18	19	19	19	19	19	**20**	**20**
SOKOBAN	3.8	**3.9**	2.9	2.9	2.9	2.9	3.8	**3.9**	2.9
	4	**4**	3	3	3	3	**4**	**4**	3
STORAGE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0	0	0	0	0	0	0	0	0
TMS	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0	0	0	0	0	0	0	0	0
TURNANDOPEN	10.1	10.0	10.8	8.8	8.6	8.6	**11.0**	8.4	8.6
	20	**20**	**20**	12	12	12	19	14	14
Overall	**99.5**	97.7	93.3	73.4	76.5	76.2	87.2	78.5	76.1
	142	138	129	91	95	93	124	110	104

Table 2. Performance of new selection strategies on IPC 2011 benchmarks. Gray rows show IPC scores, white rows coverage.

It can be seen that $\mathcal{C}$ and $\mathcal{O}$ yield very promising results with a higher coverage compared to the original method, especially in ELEVATORS and PARCPRINTER. The reason for the good performance in ELEVATORS seems to be that by narrowing the set of preferred operators the weakness of the heuristic to switch between subgoals during search can be overcome by focusing on a specific goal. In doing so, it is better to focus on the cheapest goal ($\mathcal{C}$) than on an arbitrary one ($\mathcal{O}$). It is useless, however, to focus on the most expensive goal ($\mathcal{E}$), as this changes to often during search. In PARCPRINTER both the cheapest and the most expensive goal vary a lot during search, so it is best to focus on a fixed goal like $\mathcal{O}$ does. Unfortunately, $\mathcal{O}$ does yield very bad results in CREWPLANNING, where a specific goal ordering needs to be respected that $\mathcal{O}$ is not aware of. Here, techniques to detect goal orderings [18] might be very helpful. While coverage can be increased using our new techniques, their produced solutions are typically of lower quality than those of the original method as they drive the search to satisfy goals more sequentially. This fact becomes apparent especially in OPENSTACKS, a domain for which it is very easy to find a solution but the range of quality is very high and it is important to start the right actions first in order to create concurrent solutions. Interestingly, $\mathcal{E}$ works quite well in this domain, as the actions that are needed to be started first in order to create a compact solution are also the most expensive ones.

Another interesting observation is that in the good performing methods $\mathcal{C}$ and $\mathcal{O}$ it is advantageous to concentrate on a smaller set of subgoals, while the converse holds for the poor performing method $\mathcal{E}$. This is due to the fact that with increasing size of the preferred operators set the original set is resembled more and more.

The most important observation that can be made from this experiment has motivated the design of the search procedure presented in the previous section: Different selection strategies have strengths in different domains and it appears to be very desirable to combine these strengths in a general way. Table 3 shows results of an implementation of Algorithm 1 combining narrowing strategies with a queue containing the original preferred operators ($\mathcal{P}$). The method that profits the most from this combination is $\mathcal{O}^1$, with the most important factor being the gain in CREWPLANNING. Besides, the other versions profit also, especially in terms of quality. Both $\mathcal{PC}$ and $\mathcal{PO}$ achieve higher IPC scores than $\mathcal{P}$ alone. Table 4 shows that the power of combining selections strategies can be exploited even further, with $\mathcal{PO}^1\mathcal{C}^1\mathcal{E}^1$, abbreviated as TFD$^+$ in the table, achieving both the highest coverage and IPC score.

IPC 2011	$\mathcal{PC}^1$	$\mathcal{PC}^2$	$\mathcal{PC}^3$	$\mathcal{PE}^1$	$\mathcal{PE}^2$	$\mathcal{PE}^3$	$\mathcal{PO}^1$	$\mathcal{PO}^2$	$\mathcal{PO}^3$
CREWPLANNING	**19.9**	**19.9**	**19.9**	**19.9**	**19.9**	**19.9**	**19.9**	**19.9**	**19.9**
	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
ELEVATORS	**15.1**	7.7	6.5	0.0	0.0	0.0	13.4	4.3	0.9
	18	9	7	0	0	0	**18**	6	1
FLOORTILE	4.6	**4.7**	4.5	4.4	4.5	4.3	4.5	4.5	4.4
	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
MATCHCELLAR	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**	**15.6**
	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
OPENSTACKS	17.9	18.0	18.3	18.5	**18.7**	18.6	18.0	17.7	18.1
	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
PARCPRINTER	1.0	1.0	1.8	0.0	0.0	0.0	**9.3**	0.9	0.0
	1	1	2	0	0	0	**10**	1	0
PARKING	13.9	**15.7**	13.4	10.9	11.5	11.1	11.9	12.9	12.3
	18	**20**	17	15	15	15	16	17	16
PEGSOL	17.9	17.9	18.5	17.9	18.4	18.5	**18.7**	18.5	18.6
	18	18	**19**	18	**19**	**19**	**19**	**19**	**19**
SOKOBAN	2.9	2.9	2.9	**3.0**	2.9	2.9	**3.0**	2.9	2.9
	3	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
STORAGE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0	0	0	0	0	0	0	0	0
TMS	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0	0	0	0	0	0	0	0	0
TURNANDOPEN	13.9	13.2	13.2	13.0	13.4	12.9	**14.2**	13.7	12.6
	20	19	19	18	19	18	**20**	19	18
Overall	122.6	116.6	114.5	103.1	104.8	103.8	**128.4**	110.8	105.3
	143	135	132	119	121	120	**151**	130	122

Table 3. Combining narrowing strategies with the original selection strategy ($\mathcal{P}$) via restarting as described in Algorithm 1. Gray rows show IPC scores, white rows coverage.

IPC 2011	$\mathcal{PC}^1\mathcal{O}^1$	$\mathcal{PC}^1\mathcal{E}^1$	$\mathcal{PO}^1\mathcal{E}^1$	**TFD$^+$**
CREWPLANNING	**19.9** (**20**)	**19.9** (**20**)	**19.9** (**20**)	**19.9** (**20**)
ELEVATORS	13.4 (**18**)	**15.4** (**18**)	13.4 (**18**)	13.4 (**18**)
FLOORTILE	**4.8** (**5**)	4.6 (**5**)	4.7 (**5**)	**4.8** (**5**)
MATCHCELLAR	15.6 (**20**)	15.6 (**20**)	15.6 (**20**)	**15.6** (**20**)
OPENSTACKS	17.9 (**20**)	18.2 (**20**)	**18.3** (**20**)	17.9 (**20**)
PARCPRINTER	9.5 (**10**)	0.9 (1)	9.5 (**10**)	**9.5** (**10**)
PARKING	14.1 (18)	13.8 (17)	12.0 (16)	**14.6** (**19**)
PEGSOL	**18.6** (**19**)	18.5 (**19**)	18.3 (**19**)	**18.6** (**19**)
SOKOBAN	2.9 (**3**)	**3.0** (**3**)	2.9 (**3**)	2.9 (**3**)
STORAGE	0.0 (0)	0.0 (0)	0.0 (0)	0.0 (0)
TMS	0.0 (0)	0.0 (0)	0.0 (0)	0.0 (0)
TURNANDOPEN	14.0 (**20**)	13.2 (19)	**14.1** (**20**)	14.0 (**20**)
Overall	130.7 (153)	123.1 (142)	128.7 (151)	**131.2** (**154**)

Table 4. IPC scores and coverage (in parentheses) of combining more than one narrowing strategy via restarting as described in Algorithm 1. TFD$^+$ is an abbreviation for $\mathcal{PC}^1\mathcal{O}^1\mathcal{E}^1$.

To see how these improvements affect the performance of TFD relatively to other temporal planning systems, we compared both the original TFD and TFD$^+$, a version of TFD that implements Algorithm 1 with queues for $\mathcal{P}$, $\mathcal{O}^1$, $\mathcal{C}^1$, and $\mathcal{E}^1$ to the participants of the temporal satisficing track of IPC 2011 that achieved at least one point in the competition. For this experiment, we did not re-run the other planning systems, but use the raw results of the competition directly.[4]

Table 5 presents IPC scores (gray rows) and coverage (white rows). It can be seen that TFD$^+$ clearly outperforms all competitors both in terms of coverage and IPC score.

Note that for some planners the scores presented in this paper vary from the scores they received in the competition as we did find better plans for many problems and used them as reference plans to compute all scores. For example, CPT4, which is optimal for the conservative semantics of Smith and Weld [23], produced some non-optimal plans in Floortile and Parcprinter. This was not recognized as its plans were the best of those generated during the competition.

IPC 2011	**CPT4**	**LMTD**	**YAHSP2**	**YAHSP2-mt**	**POPF2**	**DAE-YAHSP**	**TFD**	**TFD$^+$**
CREWPLANNING	7.0	0.0	16.0	15.9	**20.0**	**20.0**	19.9	19.9
	7	0	**20**	**20**	**20**	**20**	**20**	**20**
ELEVATORS	0.0	6.7	8.6	8.9	2.2	12.3	1.0	**13.4**
	0	9	**20**	**20**	3	15	1	18
FLOORTILE	**12.1**	4.8	6.9	8.3	0.0	7.3	4.9	4.8
	15	5	13	**15**	0	12	5	5
MATCHCELLAR	0.0	12.5	0.0	0.0	15.3	0.0	**15.6**	**15.6**
	0	15	0	0	**20**	0	**20**	**20**
OPENSTACKS	0.0	0.0	12.6	12.1	15.0	**19.9**	17.7	17.9
	0	0	**20**	19	**20**	**20**	**20**	**20**
PARCPRINTER	2.0	0.0	3.7	4.7	0.0	2.0	0.0	**9.5**
	5	0	7	8	0	4	0	**10**
PARKING	0.0	0.0	11.0	12.7	14.7	**15.9**	12.2	14.6
	0	0	**20**	**20**	**20**	**20**	16	19
PEGSOL	19.0	19.9	17.2	18.0	18.6	**20.0**	17.9	18.6
	19	**20**	**20**	**20**	19	**20**	18	19
SOKOBAN	0.0	0.0	10.9	**11.6**	2.5	4.5	2.9	2.9
	0	0	**12**	**12**	3	6	3	3
STORAGE	0.0	0.0	2.7	7.2	0.0	**15.5**	0.0	0.0
	0	0	5	11	0	**19**	0	0
TMS	0.0	0.0	0.0	0.0	**5.0**	0.0	0.0	0.0
	0	0	0	0	**5**	0	0	0
TURNANDOPEN	0.0	7.0	0.0	0.0	7.8	0.0	13.3	**14.0**
	0	13	0	0	9	0	**20**	**20**
Overall	40.1	50.9	89.6	99.3	101.1	117.2	105.3	**131.2**
	46	62	137	145	119	136	123	**154**

Table 5. Gray rows show IPC scores, white rows coverage of participants of IPC 2011 that solved at least one instance. The two rightmost columns show results of TFD using its original setting and using all new techniques presented in this paper (TFD$^+$).

In another experiment, presented in Table 6, we focus on quality by comparing TFD featuring our techniques, called TFD$^+$, pairwise to all other planners of IPC 2011, only considering problems where both planners have found a solution by computing the ratio between the makespan of those solutions. Scores greater than 1.0 therefore indicate that we found plans of higher quality. It can be seen that our plans offer the highest quality throughout all domains.

Finally, in our last experiment we show that the good performance of our techniques is not only a phenomenon on a specific benchmark set, but occurs on a wider range of domains. Therefore, we use the benchmark suites of IPCs 2006 and 2008 (excluding Pathways and TPP, where not only makespan but a more complex metric needs to be optimized, a feature TFD cannot deal with yet). Results are presented in Table 7. Note that only for the benchmark set of 2008 reference plans are used. TFD$^+$ has a higher coverage in all domains but PIPESWORLD. (In PIPESWORLD, h^{cea} fails to mark certain relevant operators as preferred, requiring to extract those operators from the

[4] IPC 2011 has been run on INTEL Xeon 2.93 GHz Quad Core processors with a memory limit of 6 GB and a timeout of 30 minutes. Note that TFD (like most processes) generally runs faster on such a system than on the system we used to generate our results, so the comparison is in favor of the planning systems that participated in the competition.

IPC 2011	**CPT4**	**LMTD**	**Y2**	**Y2-mt**	**POPF2**	**DAE-Y**	**TFD**
CREWPLANNING	7 **1.00**	–	20 **1.29**	20 **1.29**	20 0.99	20 0.99	20 **1.00**
ELEVATORS	–	9 0.94	18 **2.08**	18 **1.98**	3 **1.01**	15 0.93	1 0.59
FLOORTILE	5 **1.67**	2 0.99	4 **2.38**	5 **2.22**	–	4 **2.36**	5 0.96
MATCHCELLAR	–	15 **1.06**	–	–	20 **1.25**	–	20 **1.24**
OPENSTACKS	–	–	20 **1.47**	19 **1.46**	20 **1.23**	20 0.92	20 **1.04**
PARCPRINTER	2 **1.76**	–	7 **1.88**	7 **1.88**	–	4 **1.95**	–
PARKING	–	–	19 **1.50**	19 **1.35**	19 **1.12**	19 **1.11**	16 **1.03**
PEGSOL	18 0.99	19 0.98	19 **1.16**	19 **1.11**	18 **1.00**	19 0.98	18 **1.00**
SOKOBAN	–	–	3 **1.02**	3 **1.03**	2 **1.17**	3 **1.10**	3 **1.00**
STORAGE	–	–	–	–	–	–	–
TMS	–	–	–	–	–	–	–
TURN AND OPEN	–	13 **1.38**	–	–	9 0.61	–	20 **1.03**
OVERALL	32 **1.15**	58 **1.08**	110 **1.54**	110 **1.48**	111 **1.08**	104 **1.08**	123 **1.05**

Table 6. Pairwise plan quality comparisons to TFD$^+$ featuring our new techniques, namely separate queues $\mathcal{O}^1$, $\mathcal{C}^1$, and $\mathcal{E}^1$, respectively, and restarting like described in Algorithm 1. Only instances that are solved by both approaches (the small number states their number) are considered. Scores greater than 1.0 indicate that TFD$^+$ generates plans of higher quality.

regular open list. While regular TFD also uses preferred operators, it expands nodes from the regular open list more often as the number of lists is smaller than in TFD$^+$.) In this experiment the title of this paper is reflected very well: Coverage is increased drastically (from 294 to 370) while the average plan quality is not only maintained, but even slightly improved.

	TFD	**TFD+**	**Quality**
IPC 2006			
OPENSTACKS	17.5 (18)	**20.0 (20)**	18 **1.03**
PIPESWORLD	**17.7 (18)**	15.1 (16)	15 0.96
ROVERS	11.9 (12)	**16.8 (17)**	12 0.99
STORAGE	**16.7 (17)**	**16.7 (17)**	17 **1.01**
TRUCKS	13.5 (14)	**29.4 (30)**	14 **1.00**
IPC 2008			
CREWPLANNING-*strips*	**29.9 (30)**	**29.9 (30)**	30 **1.00**
ELEVATORS-*numeric*	16.7 (20)	**25.2 (30)**	20 **1.02**
ELEVATORS-*strips*	13.0 (16)	**20.8 (30)**	16 0.96
MODELTRAIN-*numeric*	1.0 (1)	**5.3 (7)**	1 **1.00**
OPENSTACKS-*adl*	27.1 **(30)**	**27.6 (30)**	30 **1.02**
OPENSTACKS-*strips*	27.1 **(30)**	**28.1 (30)**	30 **1.04**
PARCPRINTER-*strips*	9.0 (13)	**22.4 (23)**	13 **1.77**
PEGSOL-*strips*	28.3 (29)	**29.3 (30)**	29 **1.01**
SOKOBAN-*strips*	**11.9 (12)**	**11.9 (12)**	12 **1.00**
TRANSPORT-*numeric*	4.9 (6)	**11.0 (18)**	6 **1.05**
WOODWORKING-*numeric*	16.6 (28)	**21.6 (30)**	28 **1.36**
Overall	262.9 (294)	**331.1 (370)**	291 **1.08**

Table 7. The two columns in the middle show IPC scores and coverage (in parentheses) of regular TFD and TFD$^+$ on the benchmarks suites of IPC 2006 and 2008. TFD$^+$ features separate queues for $\mathcal{C}^1$, $\mathcal{O}^1$, and $\mathcal{E}^1$, as well as restarting according to Algorithm 1. The last column shows pairwise plan quality comparisons between TFD and TFD$^+$ on all instances that were solved by both approaches (the small number states their number). Scores greater than 1.0 indicate that TFD$^+$ generates plans of higher quality.

5 Conclusion

In this paper we have presented novel methods to narrow sets of preferred operators. Embedding these methods in the search framework of TFD increases its coverage at the price of quality. This drawback, however, can be overcome by utilizing a restarting strategy that is incorporated into a priority-based multi-queue best-first search framework. We have implemented these techniques and have shown empirically that combining them increases the coverage of TFD by a huge amount and preserves the average quality of the produced plans.

Future work includes incorporating goal ordering techniques to find more sophisticated orderings for $\mathcal{O}$ as well as determining additional selection strategies for preferred operators that might increase the coverage of TFD even further. While this work is motivated by the large gap between coverage and quality when searching in the space of time-stamped states, it can also be applied to classical planning and doing so is a major part of our future work.

Acknowledgments

This work was supported by the German Aerospace Center (DLR) as part of the Project “Kontiplan” (50 RA 1010).

REFERENCES

[1] Christer Bäckström and Bernhard Nebel, ‘Complexity Results for SAS$^+$ Planning’, *Computational Intelligence*, **11**, 625–655, (1996).

[2] Amanda Coles, Andrew Coles, Allan Clark, and Stephen Gilmore, ‘Cost-Sensitive Concurrent Planning under Duration Uncertainty for Service Level Agreements’, in *Proc. ICAPS 2011*, pp. 34–41, (2011).

[3] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long, ‘Forward-Chaining Partial-Order Planning’, in *Proc. ICAPS 2010*, pp. 42–49, (2010).

[4] Andrew Coles, Maria Fox, Keith Halsey, Derek Long, and Amanda Smith, ‘Managing Concurrency in Temporal Planning Using Planner-Scheduler Interaction’, *AIJ*, **173**(1), 1–44, (2009).

[5] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith, ‘Planning with Problems Requiring Temporal Coordination’, in *Proc. AAAI 2008*, pp. 892–897, (2008).

[6] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld, ‘When is Temporal Planning Really Temporal?’, in *Proc. IJCAI 2007*, pp. 1852–1859, (2007).

[7] Minh Binh Do and Subbarao Kambhampati, ‘Improving Temporal Flexibility of Position Constrained Metric Temporal Plans’, in *Proc. ICAPS 2003*, pp. 42–51, (2003).

[8] Minh Binh Do and Subbarao Kambhampati, ‘Sapa: A Multi-objective Metric Temporal Planner’, *JAIR*, **20**, 155–194, (2003).

[9] Johann Dréo, Pierre Savéant, Marc Schoenauer, and Vincent Vidal, ‘Divide-and-Evolve: The Marriage of Descartes and Darwin’, in *IPC 2011*, pp. 29–30, (2011).

[10] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger, ‘Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning’, in *Proc. ICAPS 2009*, pp. 130–137, (2009).

[11] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina, ‘Planning Through Stochastic Local Search and Temporal Action Graphs in LPG’, *JAIR*, **20**, 239–290, (2003).

[12] Malte Helmert, ‘The Fast Downward Planning System’, *JAIR*, **26**, 191–246, (2006).

[13] Malte Helmert, ‘Concise Finite-Domain Representations for PDDL Planning Tasks’, *AIJ*, **173**, 503–535, (2009).

[14] Malte Helmert and Héctor Geffner, ‘Unifying the Causal Graph and Additive Heuristics’, in *Proc. ICAPS 2008*, pp. 140–147, (2008).

[15] Jörg Hoffmann and Bernhard Nebel, ‘The FF Planning System: Fast Plan Generation Through Heuristic Search’, *JAIR*, **14**, 253–302, (2001).

[16] Chihwei Hsu and Benjamin W. Wah. The SGPlan Planning System in IPC-6, 2008.

[17] Yanmei Hu, Dunbo Cai, and Minghao Yin, ‘The LMTD Planner: On the Discovery and Utility of Precedence Constraints in Temporal Planning’, in *IPC 2011*, pp. 128–131, (2011).

[18] Jana Köhler and Jörg Hoffmann, ‘On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm’, *JAIR*, **12**, 338–386, (2000).

[19] Drew McDermott, ‘A Heuristic Estimator for Means-Ends Analysis in Planning’, in *Proc. AIPS 1996*, pp. 142–149, (1996).

[20] Drew McDermott, ‘Using Regression-Match Graphs to Control Search in Planning’, *AIJ*, **109**(1–2), 111–159, (1999).

[21] Silvia Richter and Malte Helmert, ‘Preferred Operators and Deferred Evaluation in Satisficing Planning’, in *Proc. ICAPS 2009*, pp. 273–280, (2009).

[22] Gabriele Röger and Malte Helmert, ‘The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning’, in *Proc. ICAPS 2010*, pp. 246–249, (2010).

[23] David E. Smith and Daniel S. Weld, ‘Temporal Planning with Mutual Exclusion Reasoning’, in *Proc. IJCAI 1999*, pp. 326–337, (1999).

[24] Vincent Vidal, ‘CPT4: An Optimal Temporal Planner Lost in a Planning Competition without Optimal Temporal Track’, in *IPC 2011*, pp. 25–28, (2011).

[25] Vincent Vidal, ‘YAHSP2: Keep it Simple, Stupid’, in *IPC 2011*, pp. 83–90, (2011).