ICAPS 2011 Workshop on Heuristics for Domain Independent Planning

Enhancing Search for Satisficing Temporal Planning with Objective-driven Decisions

J. Benton[†] and **Patrick Eyerich**[‡] and **Subbarao Kambhampati**[†]

[†] Dept. of Computer Science and Eng.
Arizona State University
Tempe, AZ 85287 USA
{j.benton,rao}@asu.edu

[‡] Department of Computer Science
University of Freiburg
Freiburg, Germany
eyerich@informatik.uni-freiburg.de

Abstract

Heuristic best-first search techniques have recently enjoyed ever-increasing scalability in finding satisficing solutions to a variety of automated planning problems, and temporal planning is no different. Unfortunately, achieving efficient computational performance often comes at the price of clear guidance toward solution of high quality. This fact is sharp in the case of many best-first search temporal planners, who often use a node evaluation function that is mismatched with the objective function, reducing the likelihood that plans returned will have a short makespan but increasing search performance. To help mitigate matters, we introduce a method that works to progress search on actions declared "useful" according to makespan, even when the original search may ignore the makespan value of search nodes. We study this method and show that it increases over all plan quality in most of the benchmark domains from the temporal track of the 2008 International Planning Competition.

Introduction

Heuristic best-first search planning methods have been the de facto standard for generating scalable satisficing planning algorithms for over a decade. This success has lead to using these techniques in a broad range of problems, including temporal planning. In temporal planning problems, actions have duration and the objective function (f_{OF}) is typically to minimize plan makespan (i.e., f_{OF} is plan makespan). However, as recently pointed out by Benton et al. (2010), using makespan to evaluate search nodes (i.e., in typical A^*-like fashion with $f = g + h$) can significantly reduce the scalability of search. Instead, most temporal planners tend to avoid directly searching on makespan and use other measures for node evaluation to achieve scalable performance (c.f., Sapa (Do and Kambhampati 2003) and TFD (Eyerich, Mattmüller, and Röger 2009)). Unfortunately, taking this approach can come at the cost of plan quality—one cannot necessarily expect a low makespan when the node evaluation of search is mismatched with the objective function.

Although the strategy of optimizing for f_{OF} using a search not explicitly guided by f_{OF} might seem quixotic, we have argued elsewhere (Cushing, Benton, and Kambhampati 2011) that this is, in fact, the only reasonable thing to do when there is a high cost-variance. Indeed, in makespan planning we have g-value plateaus over f_{OF} (i.e., where g_{OF} does not change from the parent to child node and $f_{OF} = g_{OF} + h_{OF}$) and hence an implied infinite cost variance. Nevertheless, while strategies that use a node evaluation function other than f_{OF} can improve scalability, and may find reasonably low makespan plans using branch-and-bound search, the quality improvement within a given time limit may not be sufficiently high.

To try to improve matters, we adopt a general way of pushing the base-level search toward higher quality plans according to f_{OF}. We seek to augment search approaches where $f \neq f_{OF}$ and f is the node evaluation function for an A^*-like best-first search. We adapt the concept of "uselessness" introduced by Wehrle, Kupferschmid, and Podelski (2008), who prune away certain states from consideration that appear to offer no benefit for finding a solution toward the goal. In specific, our approach evaluates search nodes according to their degree of "usefulness" on f_{OF} itself.

The idea is to provide a window into the objective function while still maintaining scalability. This is done, in particular, on areas of the search space that appear to have *zero-cost* according to f_{OF}. For makespan, this lets us consider parallelizing actions as much as possible (i.e., adding an action that runs in parallel with others often has no effect on makespan, so it is highly desirable for minimization). Among nodes generated from a *zero-cost* (i.e., *zero-makespan*) action, we choose the node with high usefulness for immediate expansion, a local lookahead that allows the main search algorithm to then evaluate its children on the evaluation function f. In the end, we get a final heuristic search that proceeds in two steps: (1) a best-first search step that uses the planner's usual *open list* structure and search (i.e., including its usual node evaluation function f used by the search), and (2) a local decision step where the most useful search states (according to f_{OF}) are expanded.

The rest of this paper proceeds as follows. After an overview of background and notation, we discuss the idea of operator usefulness. We then show how to embed local decisions on usefulness into a best-first search framework. Finally, we present empirical analysis showing the quality benefits that can be gained by using this approach in the planner Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009), a state-of-the-art best-first search heuristic planner for temporal planning.

Background and Notation

Before going into our technique, we must first cover some background information and notation on best-first search, temporal planning, g-value plateaus and useless search operators.

Best-First Search

Best-first search is a well-known algorithm for searching over a (possibly infinite) state space from an initial state I to a state that entails the goal G (Pearl 1984). We provide a brief overview of it while introducing notation. The algorithm, shown in Algorithm 1, provides an *open list*, where the best-valued state may be removed, and a *closed list* which allows for duplicate detection.

It begins with the initial state I in the open list. It then repeats the following process: The best-valued state (according to an evaluation function f) s is taken. If s is not in the closed list it is added, otherwise we skip this state (as it has already been explored). If $s \models G$, then s is returned as a solution, otherwise s is *expanded*. This means all possible successor states (also known as *child* states) are generated. To generate a child state we use an operator $o = \langle p, e, c \rangle$, where p is a set of conditions for the applicability of o, e is a transformation function on states (i.e., effects), and c is a constant-valued cost. o is considered applicable on s (the *parent* state) if $s \models p$. Provided the conditions of o are met, we define $o(s) = s'$ to be the child state of s, such that s' is the state resulting from the transformation function e on s. Each child state is evaluated according to f. In the best-first search algorithm A^*, $f(s') = g(s') + h(s')$, where g is a function returning the current (known) cost of s' and h is a function returning a heuristic distance from s' to the least-cost goal node. After evaluation, s' is added into the open list.

Importantly, the function f and its components g and h, are often defined in terms of the objective function of the search. For instance, in cost-based classical planning, where we seek to minimize the cost of a plan, we can define f, g, and h in terms of the summed costs of actions in a plan. Provided h always returns a lower bound on the cost to a goal, this guarantees optimal solutions. Even when this is not the case, the conventional wisdom and empirical evidence points to this yielding better-quality solutions than metrics like search distance (c.f., Richter and Westphal (2010)).

Algorithm 1: Best-first search.

```
1  open.add(I)
2  closed ← ∅
3  while open is not empty do
4      s ← open.remove_best_f()
5      if s ∉ closed then
6          closed.add(s)
7          if s ⊨ G then
8              return s as solution
9          forall the child states s' of s do
10             if s' is no dead-end then
11                 open.add(s')
12 return no solution found
```

Temporal Planning

In temporal planning, actions can run concurrently with one another and have duration, and the (typical) objective is to minimize plan makespan. For simplicity, we assume a temporal semantics similar to Temporal GraphPlan (TGP) (Smith and Weld 1999). That is, action conditions must hold through their entire duration and effects can only change at the end of an action (i.e., they cannot be changed by some other action running in parallel).

For our discussion, we assume planners with a forward state-space search space like that of Sapa (Do and Kambhampati 2003) and Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009), where each state has a time stamp t that defines what point in time actions are added and an event queue (or agenda), that defines specific *decision epochs* where actions end. A special search operation called *advance time* increases t from its current value to the next (i.e., smallest) decision epoch and applies the effects of the action at that time point.[1]

Best-first search satisficing temporal planners use a range of techniques to speed up search. We cannot go over all of those here. It is important, however, to understand how these planners might go about minimizing makespan. We are concerned with two related questions (1) how to define the cost of search operations and (2) how to define g-values and h-values. To answer (1), we consider the objective function (i.e., makespan minimization). Any search operation that adds to makespan has a cost equal to the amount of makespan added, all others do not. From this decision, it follows that we should define g as the makespan of the partial state, g_m, and h as the remaining "makespan-to-go", h_m. This makes $f_m = f_{OF}$ (where we introduced f_{OF} as the objective function in the introduction).

Some planners consider g to be the time stamp of a state, which we call g_t, and the remaining makespan from that point to be estimated by a heuristic h_t (Haslum and Geffner 2001). When summed, these turn out to be equal values, i.e., $g_t + h_t = g_m + h_m$. Other planners (such as Sapa (Do and Kambhampati 2003)) have an evaluation function that does not return makespan values.[2] Since no known comprehensive studies have been conducted on various definitions of f for temporal planning, it is difficult to say what effect this has on final plan makespan. However, as we pointed out earlier, some evidence in cost-based planning points to a negative effect (Richter and Westphal 2010).

g-value Plateaus

A g-value plateau is a region of the search space where the evaluation of states with respect to g does not change from parent to child. In state-space temporal planners, as highlighted by Benton et al. (2010), g-values can occur in abun-

[1] Note that some forward state-space temporal planners do not use this exact method for search (c.f., Coles et al. (2009)), but we frame our discussion in this way since the planner we use for our experiments (Temporal Fast Downward) does.

[2] Also note that the Temporal Fast Downward planner uses g_t as its g-value, and a heuristic on the sum of action durations.

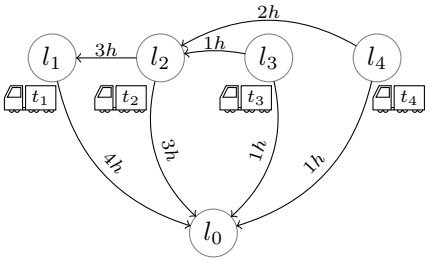


Figure 1: Layout for the example problem.

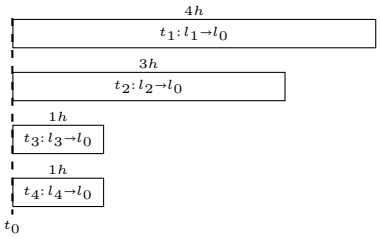


Figure 2: A makespan-optimal plan for the example.

dance and can cause problems for best-first search. To see this, let us look at an example.

Example: Consider the simple problem shown in Figure 1 of moving trucks in a logistics domain. We have 5 locations $\{l_0, \ldots, l_4\}$ all connected in the following way: Driving from location l_1 to l_0 takes 4 hours, from l_2 to l_0 takes 3 hours, and from l_3 or l_4 to l_0 takes 1 hour. We can also drive from l_4 to l_2 in 2 hours, from l_3 to l_2 in 1 hour, and from l_2 to l_1 in 3 hours. Trucks $t_1, \ldots, t_4$ start respectively at $l_1, \ldots, l_4$. The goal is to move all the trucks to l_0. Each *drive* action has the condition that the truck must be at the appropriate location and the effect that the truck is at its destination. An optimal solution to this planning problem is shown in Figure 2.

Given that every optimal solution requires that, at time t_0, we begin driving t_1 from l_1 to l_0, let us assume a planner is expanding the search state with this action. In this case, every child that drives one of the other trucks (and the advance time operator) has an equal g-value. In fact, we would end up with a fairly expansive g-value (and f-value) plateau where we must try every possible way of adding the various *drive* actions across all decision epochs until we reach a plan with (an optimal) makespan of 4 hours.

Useless Search Operations

Useless operators are defined by Wehrle, Kupferschmid, and Podelski (2008) as those operators that are unnecessary for finding optimal solutions from a state s. In sequential classical planning with unit costs, you can further formalize the notion of useless operators by taking $d(s)$ as the optimal distance to a goal state from s and $d^{\bar{o}}(s)$ as the optimal distance to a goal state without an operator o (i.e., $d(s)$ is a perfect heuristic and $d^{\bar{o}}(s)$ is a perfect heuristic without the operator o). If $d^{\bar{o}}(s) \leq d(o(s))$ then o is a useless operator. To understand why consider that if $d^{\bar{o}}(s) \leq d(o(s))$, then it must also be the case that $d(s) \leq d(o(s))$. Since this is true if and only if an there are no optimal plans that start from s, o must be useless.

Unfortunately, this definition is not easily transferable to planning for makespan minimization (at least not in planners like TFD) where we have zero-cost search operations. Using the original definition, search operations would be considered useless when they were not (e.g., when adding a useful operator that runs parallel to an optimal plan we would get no increase in makespan). Instead, we can give a weaker criterion for uselessness.

Definition 1. *An operator o is* guaranteed *makespan-useless when $d^{\bar{o}}_m(s) < d_m(o(s))$, where $d_m(s)$ returns the remaining makespan of a plan.*

This definition lets some useless search operations to fall through the cracks, but it catches operators that will strictly increase makespan when added to the plan.

This idea can be extended to any heuristic, such that $h^{\bar{o}}(s)$ represents running the heuristic without the operator o from state s. Wehrle, Kupferschmid, and Podelski (2008) call operators declared useless by $h^{\bar{o}}(s) \leq h(o(s))$ *relatively useless operators* in the sequential classical planning setting. We use $h^{\bar{o}}_m(s) < h_m(o(s))$ to define possibly *relatively makespan-useless* operators.

Useful Search Operations

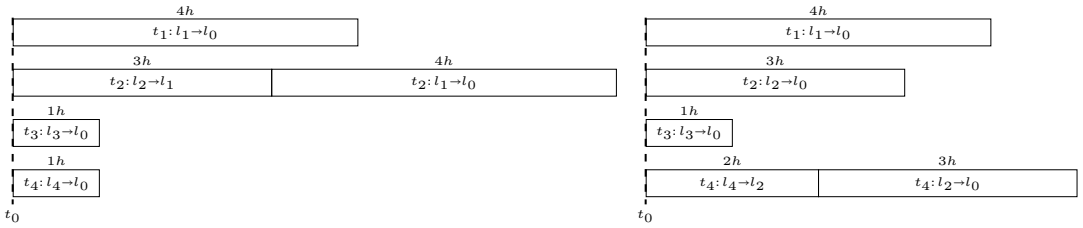
In contrast with a useless search operator, which should never be explored, a *useful* search operator is one whose absence would lead to worse plan quality. We can find a useful operator by using the same method for finding useless operators. In other words, operator o is *useful* in a state s if $d^{\bar{o}}(s) > d(o(s))$. We say the state $o(s) = s'$ is useful if the operator generating it is useful. In this section we discuss how we can use this notion to enhance satisficing planners using best-first search.

Integrating Usefulness with Satisficing Planning

To see how we might use the concept of operator usefulness, let us revisit the example from Figure 1. Every optimal plan will need to include the operator that drives t_1 from l_1 to l_0, starting from time 0. Therefore, as before, we optimistically assume that the search algorithm finds (and chooses to expand) a state s_e that already includes this operator. Otherwise we make no assumptions on the definition of g- or h-values or what order states would be expanded in during a planner's regular (perhaps modified) best-first search.

Consider the possibility of using operator usefulness here. Let us assume for a moment that we can find d_m and $d^{\bar{o}}_m$ (i.e., the perfect heuristic on makespan). We want to find *guaranteed makespan-useful* operators, or operators o where $d^{\bar{o}}_m(s) > d_m(o(s))$. In our example on the state s_e, we would find the operators depicted in Figure 3 as guaranteed makespan-useful. Again, we cannot say whether an operator whose absence does not change makespan is useful or useless.

Since the guaranteed makespan-useful search operations are within a makespan g-value plateau, they are locally without cost according to the makespan objective. We would



(a) Without t_2: $l_2 \to l_0$ (b) Without t_4: $l_4 \to l_0$

Figure 3: Plans resulting from removing useful operators.

like to greedily seize the opportunity to expand the best possible search operator, keeping in mind that, since we have no guarantees on search behavior, the possibility of expanding a certain node may never present itself again. The idea is to generate a short "tunnel" in areas of original best-first search that provide zero-cost search operations according to the objective function (which is not necessarily the evaluation function used for nodes).

Notice that some operators are more useful than others. For instance, if we are disallowed from driving t_2 from l_2 to l_0, the makespan of the plan increases by 3 hours. In contrast, if we cannot drive t_4 from l_4 to l_0 the makespan increases by just 1 hour. Looking at Figure 3 again, indeed driving t_2 from l_2 to l_0 is the most useful operator, as we lose the most time when we do not include it. Therefore, we would want to choose this operator before all others to include in the plan.

More formally, at a state s, we can define the makespan *heuristic degree of usefulness* of an operator o as $\upsilon^o(s) = h^{\bar{o}}_m(s) - h_m(o(s))$. With this information, we can create a new search algorithm that interleaves local search decisions using degree of usefulness on makespan g-value plateaus (i.e., plateaus on g_m) and any best-first search strategy. We show how the interleaving would work when combined with a best-first search strategy in Algorithm 2

Up to line 12, the algorithm follows regular best-first search. At that point, generated child states are split from the regular search and placed into a υ*list* if they are on a g_m-value plateau and are possibly *makespan-useful*. Otherwise, they are placed in the regular *open* list. On line 18, the algorithm begins the local decision stage. If there are any nodes in the υ*list* (and there exists at least one node that has a value unequal to the others), the most useful node is removed and expanded, its child states placed directly into the *open* list. Finally, the rest of the υ*list* contents are placed into the *open* list. On the next iteration on line 4, the υ*list* is cleared.

Algorithm 2: Local decisions on degree of usefulness interleaved with best-first search for temporal planning.

```
1  open.add(I)
2  closed ← ∅
3  while open is not empty do
4      υlist ← ∅
5      s ← open.remove_best_f()
6      if s ∉ closed then
7          closed.add(s)
8          if s ⊨ G then
9              return s as solution
10         forall the child states s′ of s do
11             if s′ is not dead-end then
12                 if g_m(s) = g_m(s′) and υ^o(s) ≤ 0 then
13                     υlist.add(s′)
14                 else
15                     open.add(s′)
16         if υlist is not empty &
17           ∃s_1, s_2 ∈ υlist: o_1(s) = s_1,
18            o_2(s) = s_2 & υ^{o_1}(s) < υ^{o_2}(s) then
19            s″ ← υlist.remove_best_υ^o
20            closed.add(s″)
21            if s ⊨ G then
22                return s as solution
23            forall the child states s‴ of s″ do
24                if s‴ is not dead-end then
25                    open.add(s‴)
26         open ← open ∪ υlist
27 return no solution found
```

Evaluation

We have shown how to integrate our strategy for making local decisions into best-first search. In this section, we evaluate this approach in the planner Temporal Fast Downward (TFD) (Eyerich, Mattmüller, and Röger 2009). The best-first search in TFD uses a modified version of the context-enhanced additive heuristic (Helmert and Geffner 2008) that sums the durations as operator costs, meaning the heuristic captures a sequential view of actions. With this heuristic, h^{cea}_{dur}, the planner uses a state's timestamp as its g-value (i.e., g_t) for a final $f = g_t + h^{cea}_{dur}$. Its search strategy uses preferred operators as defined in its classical planning counterpart, Fast Downward (Richter and Helmert 2009). TFD is an *anytime* planner, searching until it exhausts the search space or it reaches a timeout limit. It also reschedules solutions after it finds them in an attempt to minimize makespan (so a sequential plan may be rescheduled to a highly parallel one).

To detect possibly makespan-useful operators, we used the heuristic produced by Benton et al. (2010) for TFD in their original study of g-value plateaus. It uses a modified version of h^{cea}_{dur} to capture causal and temporal constraints between actions, detected during the extraction of a relaxed

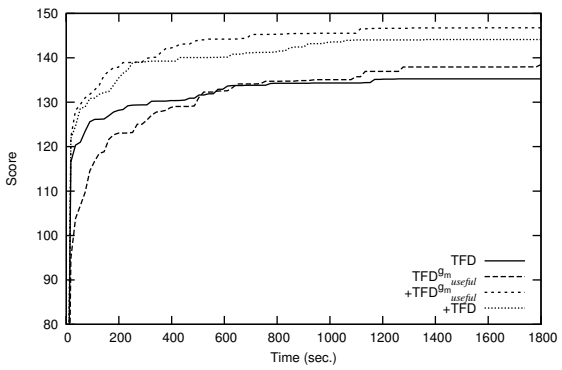


Figure 4: Anytime behavior showing the change in IPC score as time progresses.

plan. These constraints are encoded in a Simple Temporal Network (STN). The makespan of the schedule produced by the solution to the STN is returned as a heuristic estimate.

We implemented Algorithm 2 into TFD for a planner we call TFD$^{g_m}_{useful}$. Additionally, we generated a two-phase version of the planner. This variant employs *lazy evaluation* in the original TFD planner in a first phase. This means each state's parent heuristic is used (rather than its own value). After a solution is found (or 8 minutes has passed, whichever comes first) the planner restarts into a second phase using TFD$^{g_m}_{useful}$. It turns out that, using lazy evaluation, TFD tends to find (lower quality) solutions fast. The search uses the plan found in the first phase (if any) to prune the search space of the second phase (which focuses on finding higher quality plans using the useful actions technique). This is similar to the strategy employed by LAMA, though that planner restarts with different weights in a weighted A* search (Richter and Westphal 2010). We call this version +TFD$^{g_m}_{useful}$. Finally, to help discover the benefits granted by the two-phased search, we implemented +TFD, which uses TFD with lazy evaluation until a solution is found (or 8 minutes has passed) then switches to TFD without lazy evaluation.

Along with the original TFD, we tested all these variants on the temporal domains from the 2008 International Planning Competition (IPC-2008) (except *modeltrain*, as in our tests TFD is unable to solve more than 2 problems in this domain, depending on its settings). This benchmark set is known to contain plateaus on g_m (Benton et al. 2010).[3] The experiments were run on a 2.7 GHz AMD Opteron processor, with a timeout of 30 minutes and a memory limit of 2 GB.

Recall that, to find whether a search operator is useful requires a heuristic to be run on both the parent (without considering the operator) and the child (using the normal heuristic). Let us analyze about how many times we are running heuristics in TFD$^{g_m}_{useful}$ and the second phase of +TFD$^{g_m}_{useful}$ as compared to the original TFD. Assume n nodes are generated by the search with e nodes expanded and a branching factor of b. We will always run h^{cea}_{dur} on every child node, so TFD would run this heuristic n times. Assuming a percentage of states in g-value plateaus, r, we must run the modified (and slower) makespan heuristic in TFD$^{g_m}_{useful}$ about $b \times e \times r + n \times r$ times. Therefore, we may expect a slowdown in finding solutions from TFD to TFD$^{g_m}_{useful}$.

[3] We used the *elevators-numeric* and *openstacks-adl* variants for our results on the respective domains following the IPC-2008 rules by using the domain variant that all our planner versions did best in.

Domain	TFD	+TFD	TFD$^{g_m}_{useful}$	+TFD$^{g_m}_{useful}$
Crewplanning	22.44	29.64	22.57	29.66
Elevators	13.88	14.16	15.40	15.79
Openstacks	25.79	27.22	27.81	28.78
Parcprinter	8.73	8.74	8.47	8.50
Pegsol	28.73	28.73	28.81	28.81
Sokoban	10.93	10.88	10.79	10.79
Transport	5.06	5.06	4.68	4.68
Woodworking	19.72	19.71	19.93	19.73
Overall	**135.28**	**144.13**	**138.48**	**146.75**

Table 1: Results using the measure from IPC 2008.

Domain	TFD	+TFD	TFD$^{g_m}_{useful}$
Crewplanning	(24) 8%	(30) 0%	(24) 7%
Elevators	(19) 7%	(19) 6%	(20) 3%
Openstacks	(30) 13%	(30) 6%	(30) 4%
Parcprinter	(13) -4%	(13) -4%	(13) 1%
Pegsol	(30) 0%	(30) 0%	(30) 0%
Sokoban	(11) -1%	(11) -1%	(11) 0%
Transport	(6) 9%	(6) 9%	(6) 0%
Woodworking	(26) 7%	(26) 7%	(25) -1%
Total	(159) **6%**	(165) **3%**	(159) **3%**

Table 2: Percentage improvement made by +TFD$^{g_m}_{useful}$, comparing plans solved by both approaches (number of plans solved by both approaches shown in parenthesis).

To help see whether this greatly effects the search, we measured the quality and coverage across all benchmarks over time using the scoring system from the IPC-2008, which captures a notion of coverage and quality by summing the values of all $Q*/Q$, where $Q*$ is the makespan of the best known solution for a problem and Q is the makespan found by the planner for the problem (unsolved instances get a score of 0). The results of this appear in Figure 4. As expected, the scores for TFD$^{g_m}_{useful}$ start lower than TFD, but at about 10 minutes dominate it. On the other hand, +TFD$^{g_m}_{useful}$ dominates +TFD very quickly. Given that +TFD and +TFD$^{g_m}_{useful}$ use the same initial search (i.e., TFD with lazy evaluation), this provides an indication that the *useful actions* lookahead technique used in combination with the two-phase search is more effective than applying the useful actions technique alone. Hence, with a computational time limit we have a greater likelihood of finding a better solution if we stop +TFD$^{g_m}_{useful}$ than +TFD (and after around 10 minutes, the same holds true for TFD$^{g_m}_{useful}$ vs. TFD).

The final IPC scores for all four variations are found in Table 1. Notice that +TFD$^{g_m}_{useful}$ is better than the other approaches across problems. Looking carefully across the do-

mains, we can see the individual increases in IPC score are fairly minor between TFD$^{g_m}_{useful}$ and +TFD$^{g_m}_{useful}$ in all but the *Crewplanning* domain, for which the lazy heuristic evaluation scheme tends to work quite well. +TFD shows that this improvement appears to come from using lazy evaluation for finding a first solution. As far as the IPC metric is concerned, we get better values when using useful actions (i.e., between TFD vs. TFD$^{g_m}_{useful}$ and +TFD vs. +TFD$^{g_m}_{useful}$) in *Crewplanning*, *Elevators*, *Openstacks*, *Pegsol*, and *Woodworking*.

To get a better view of the plan-for-plan quality improvements, we compare direct makespan decreases (i.e., quality increases) on the problems solved by TFD, +TFD, TFD$^{g_m}_{useful}$ and +TFD$^{g_m}_{useful}$. Table 2 shows the percentage improvement on each domain as compared with +TFD$^{g_m}_{useful}$ for problems solved by both planners. It is evident that +TFD$^{g_m}_{useful}$ gives better quality plans over the usual TFD search in most of the domains (all but *Parcprinter* and *Sokoban*, for which there is a small decrease in plan quality). We get the largest increase in *Openstacks*, a domain where sequential plans are often easy to find (at least with TFD) but less easy to reschedule into parallel ones.

Related Work

For this work, we sought to give some degree of remedy for maintaining quality while employing scalability techniques for searching over zero-cost search operators. As mentioned earlier, others have noticed this problem. Richter and Westphal (2010) provide an investigation into the issue of variable-cost operators in the best-first search classical planner LAMA, where they notice better scalability (but lower quality) when ignoring search operator costs completely versus taking costs into account. The work most related to ours was done by Benton et al. (2010), where they study the problem of g-value plateaus that occur in temporal planning in-depth. Cushing, Benton, and Kambhampati (2011) also study the broader problem of searching using variable-cost operators, while Thayer and Ruml (2009) propose a search method for bounded-suboptimal search with variable operator costs.

Our approach for tackling the problem is related to two common enhancements to best-first search for satisficing planning: (1) methods that permanently or temporarily prune search spaces using heuristics and (2) methods that perform lookaheads. The prime examples of pruning search using heuristics are *helpful actions* (Hoffmann and Nebel 2001) and *preferred operators* (Richter and Helmert 2009). These techniques find a relaxed solution to a planning problem and exclude all operators that fail to achieve values which are part of it, marking included operators as helpful or preferred.

Interestingly, the idea of *helpful actions* was extended upon to generate a lookahead technique in the planner YAHSP (Vidal 2004). This planner attempts to use the entire relaxed solution to perform a lookahead. That is, it tries to use each action in the relaxed plan, applying them in order to reach deeper into the search space. It turns out that this approach works quite well to increase solution coverage across benchmarks, but does not do as well in terms of the quality of the solutions found. Others have expanded on this idea by attempting to find relaxed plans that would perform better for lookahead (Baier and Botea 2009), and still others have used a more of a local search approach for probing the search space based on solutions to relaxed plans (Lipovetzky and Geffner 2011).

Finding whether an operator is useful or useless also has connections to landmarks (Hoffmann, Porteous, and Sebastia 2004). It requires that we run the heuristic without an operator o at state s. For most reasonable heuristics, if we get a value of *infinity* when excluding o from its computation, then o is a landmark to finding a solution from s.

Summary and Future Work

We have shown a novel method for enhancing heuristic best-first search for satisficing planning. Our approach focuses on finding plans of higher quality by exploiting the notion of *usefulness* on the objective function f_{OF}. The idea is to turn the search toward solutions of higher quality, even when the node evaluation function differs from f_{OF}. In particular, our technique performs a lookahead on useful operators over g-value plateaus in temporal planning, where the objective is to minimize makespan. We applied this method to the best-first search of the planner Temporal Fast Downward (TFD). Our experiments on the variants TFD$^{g_m}_{useful}$ and +TFD$^{g_m}_{useful}$ show that using this method, we can achieve better quality solutions.

For future work, we are investigating how to apply similar approaches to cost-based classical planning, where often the biggest problem is not g-value plateaus, but variable cost operators. Further, we are considering ways of exploiting the connection to *local* landmarks to enhance the search further.

Acknowledgements We thank Robert Mattmüller for discussions on this work. This research is supported in part by ONR grants N00014-09-1-0017 and N00014-07-1-1049, the NSF grant IIS-0905672, and by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037.

References

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In *Proc. ICAPS 2009*.

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-value plateaus: A challenge for planning. In *Proc. ICAPS 2010*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. Extending the use of inference in temporal planning as forwards search. In *ICAPS*.

Cushing, W.; Benton, J.; and Kambhampati, S. 2011. Cost-based satisficing search considered harmful. In *Proceedings of the Workshop on Heuristics in Domain-independent Planning*.

Do, M. B., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *JAIR* 20:155–194.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. ICAPS 2009*, 130–137.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. ECP 2001*, 121–132.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proc. ICAPS 2008*, 140–147.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *ICAPS*.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009*, 273–280.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI 1999*, 326–337.

Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *ICAPS*.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proc. ICAPS 2004*, 150–159.

Wehrle, M.; Kupferschmid, S.; and Podelski, A. 2008. Useless actions are useful. In *Proc. ICAPS 2008*, 388–395.