Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning

Patrick Eyerich and **Robert Mattmüller** and **Gabriele Röger**
University of Freiburg, Germany
{eyerich,mattmuel,roeger}@informatik.uni-freiburg.de

Abstract

Planning systems for real-world applications need the ability to handle concurrency and numeric fluents. Nevertheless, the predominant approach to cope with concurrency followed by the most successful participants in the latest International Planning Competitions (IPC) is still to find a sequential plan that is rescheduled in a post-processing step. We present Temporal Fast Downward (TFD), a planning system for temporal problems that is capable of finding low-makespan plans by performing a heuristic search in a temporal search space. We show how the context-enhanced additive heuristic can be successfully used for temporal planning and how it can be extended to numeric fluents. TFD often produces plans of high quality and, evaluated according to the rating scheme of the last IPC, outperforms all state-of-the-art temporal planning systems.

Introduction

In contrast to classical planning, which takes only *causal* dependencies between actions into account, temporal planning also covers *temporal* dependencies and admits plans with concurrent durative actions. Another natural step towards real-world applications is the introduction of *numeric resources*. Since the aspects of planning (*which* actions should be executed) and scheduling (*when* should they be executed) can, under certain conditions, be considered as independent problems and consequently be tackled at different stages of the overall planning process, a straightforward approach to temporal planning is to separate the planning and scheduling phases. This approach is taken by SGPlan (Hsu and Wah 2008), the winner of the IPC 2006 and 2008 temporal tracks. Basically, SGPlan partitions a planning problem by parallel decomposition into loosely coupled subproblems and solves them using a variant of the well-known planning system Metric-FF (Hoffmann 2003). Only then, scheduling takes place.

Alternative approaches that integrate the planning and scheduling aspects more tightly can lead to a shorter makespan, because they also consider plans that cannot be rescheduled into sequential solutions: One such planning system is LPG (Gerevini, Saetti, and Serina 2008) which is based on local search and planning graphs. The search space of LPG consists of "action graphs", which are subgraphs of the planning graph representing partial plans. A heuristic is used to estimate the "search cost" and the "execution cost" of conditions. Crikey (Coles et al. 2009) links planning and scheduling algorithms into a planner which is capable of solving many problems with required concurrency (Cushing et al. 2007). Alternatively, one can build domain-independent heuristic forward chaining planners that can handle durative actions and numeric variables. Both TFD and Sapa (Do and Kambhampati 2003) follow this approach. While TFD and Sapa have much in common, the main difference lies in the heuristic: Sapa utilizes a temporal version of planning graphs first introduced in TGP (Smith and Weld 1999) to compute heuristic values of time-stamped states.

We show how the context-enhanced additive heuristic (Helmert and Geffner 2008) can be successfully adapted to temporal and numeric planning. The overall performance of the whole system turns out to be superior to the state of the art. Especially the *quality* of the generated plans is very high. On top of that, we are able to find plans requiring some limited form of concurrency.

The rest of this paper is structured as follows: After introducing the temporal planning formalism and the search, we briefly present the original context-enhanced additive heuristic. The following section answers the question how this heuristic can be used for temporal and numeric planning, followed by a presentation of the experimental results. We close with a discussion and an outlook on future work.

Temporal Planning Tasks

We start with the introduction of our running example, depicted in Fig. 1. The figure shows four locations ℓ_0 through ℓ_3 (the labels d_{jk} at the edges represent the distances between the locations). Two gardening robots r_1 and r_2, both initially located at ℓ_0, have to water flowers f_j at locations ℓ_j that have current water levels h_j and that need to be watered until levels n_j are reached ($j = 1, 2, 3$). Location ℓ_0 is equipped with an infinite water reservoir where the robots' water tanks can be refilled. Both tanks have capacities of $c_i = 150$ units, $i = 1, 2$, and are initially empty ($w_i = 0$).

The actions available to the robots are depicted below. The operators have start conditions (top left of the action bar) that must be satisfied at the beginning of the execution and some have persistent conditions (above the action bar) that must be satisfied during the execution (on an open interval). This example does not contain end conditions, but in

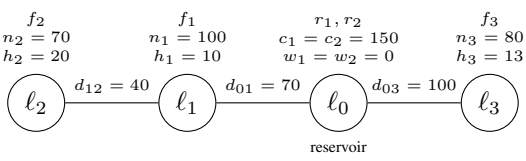
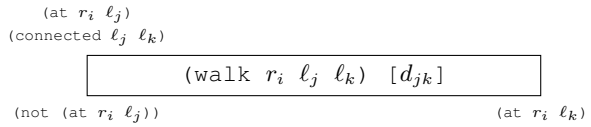


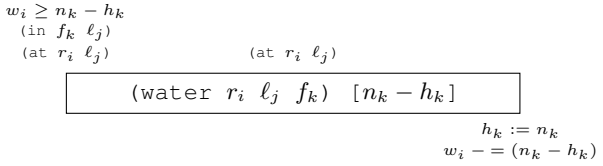
Figure 1: Gardening robots example.

general they are possible as well. Actions can have start (at the bottom left) and end effects (bottom right).

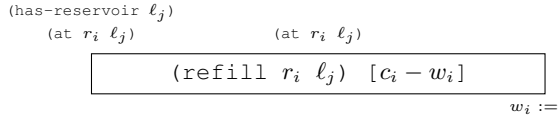
- `(walk` r_i ℓ_j ℓ_k`)`: Walking from one location to an adjacent one, with duration d_{jk}.



- `(water` r_i ℓ_j f_k`)`: Watering a flower at a certain location, with duration $n_k - h_k$.



- `(refill` r_i ℓ_j`)`: Refilling the water tank, with duration $c_i - w_i$.



One possible concurrent plan for the gardening problem is given in Fig. 2.

For our approach, we do not use the PDDL formulations of the planning instances directly, but automatically translate them to a new formalism that we call *temporal numeric SAS*$^+$ and which is based on *SAS*$^+$(Bäckström and Nebel 1995) and Helmert's finite-domain representation (2009), respectively. The main differences from PDDL are the use of multi-valued state variables and the handling of logical dependencies and arithmetic subterms via axioms. The values of these numeric variables are set directly by the actions or, in the case of compound expressions, are determined by newly introduced numeric axioms. Comparisons between numeric expressions are translated to logical variables whose values are determined by comparison axioms. Formally, we can define a *temporal numeric SAS*$^+$ *planning task* as a tuple $\Pi = \langle \mathcal{V}, s_0, s_\star, \mathcal{A}, \mathcal{O} \rangle$ with the following components:

$\mathcal{V}$ is a set of *state variables* v, partitioned into a set $\mathcal{V}_l$ of *logical variables* with finite domains $\mathcal{D}_v$ and a set of *numeric variables* $\mathcal{V}_n$ with domains $\mathcal{D}_n = \mathbb{R} \cup \{\perp\}$ ($\perp$ for *undefined*). A subset $\mathcal{V}_c \subseteq \mathcal{V}_l$ of the logical variables is distinguished as *comparison variables* with possible values *false*, *true*, and $\perp$. State variables are partitioned into *fluents* (affected by operators) and *derived variables* (computed by evaluating axioms). Each derived logical variable v has a *default value* $\mathrm{def}(v) \in \mathcal{D}_v$. 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 is defined by a partial state $s_\star$ over the logical variables. A *partial state* s' is a state restricted to a subset of the fluents. We write $\mathrm{dom}(s')$ for the subset of $\mathcal{V}$ on which s' is defined. Analogously to the Boolean setting, we sometimes 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.

$\mathcal{A} = \mathcal{A}_n \dot\cup \mathcal{A}_c \dot\cup \mathcal{A}_l$ is the set of *axioms*. $\mathcal{A}_n$ is the set of *numeric axioms*, each being of the form $v_1 = v_2 \circ v_3$, where v_1 is a derived numeric variable called the *affected variable*, $\circ \in \{+, -, *, /\}$ is the *axiom operator*, and v_2 and v_3 are the numeric *body variables*. The numeric axioms induce a *numeric dependency graph* $\mathcal{G}_n = \langle \mathcal{V}_n, \mathcal{E}_n \rangle$ with edges from v_2 and v_3 to v_1 for each numeric axiom $v_1 = v_2 \circ v_3$. We require that $\mathcal{G}_n$ is acyclic and that for each derived numeric variable v, there is exactly one numeric axiom in $\mathcal{A}_n$ with affected variable v.

The *comparison axioms* in $\mathcal{A}_c$ compare the values of numeric variables to 0 and are of the general form $v_1 = v_2 \bowtie 0$, where the *affected variable* v_1 is a comparison variable, $\bowtie \in \{<, \leq, =, \geq, >\}$ is the *comparator* and v_2 is the *body variable*. The *derived value* for v_1 is *true* if $s(v_2) \bowtie 0$ and *false*, otherwise. We require that for each comparison variable v, there is exactly one comparison axiom in $\mathcal{A}_c$ with affected variable v.

For an example of numeric and comparison axioms, see Fig. 3, illustrating the auxiliary variables introduced in order to represent the precondition $w_1 \geq n_1 - h_1$ of the durative action `(water` r_1 ℓ_1 f_1`)` from the previous example. In order to encapsulate the evaluation of numeric (sub)terms and comparisons, we introduce new auxiliary variables for them. The subgraph rooted at aux_2 is (a subgraph of) the numeric dependency graph induced by the problem. Note that using this graph allows sharing of common subterms. Hence, e.g., the expression $n_1 - h_1$ represented by aux_1 only has to be evaluated once in each state, although it is used not only in the precondition of `(water` r_1 ℓ_1 f_1`)`, but also in its effect.

Finally, $\mathcal{A}_l$ contains the *logical axioms*, which are of the form $c \rightarrow v = w$, where v is a derived logical variable. If the *condition* c, which is a partial variable assignment over $\mathcal{V}_l$, is satisfied, the axiom triggers and the *affected variable* v is assigned the *derived value* w. If no axiom affecting the variable triggers, the variable is assigned its default

The condition holds iff $w_1 - (n_1 - h_1) \geq 0$.

$$aux_1 = n_1 - h_1$$
$$aux_2 = w_1 - aux_1$$
$$aux_3 = (aux_2 \geq 0)$$

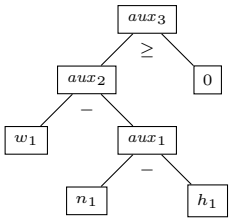


Figure 3: Visualization of numeric and comparison axioms.

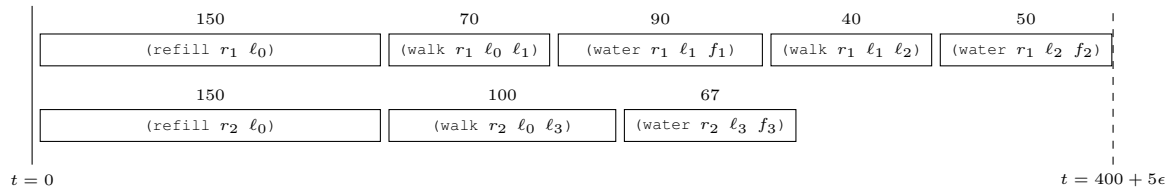


Figure 2: Plan for the gardening robots problem, with actions separated by $\epsilon > 0$, and durations written above the actions.

value. We require that $\mathcal{A}_l$ can be stratified. Arithmetic operations or comparisons involving the undefined value $\perp$ always yield $\perp$ as the derived value of the affected variable.

When axioms are evaluated, the numeric axioms have to be considered first. Traversing the numeric dependency graph "bottom-up" (starting from the nodes with in-degree 0, which hold numeric fluents), we can assign the correct value to each numeric variable. Subsequently, the comparison variables can be assigned their values in the obvious way. Last, the logical axioms are evaluated layer by layer, with one fixpoint iteration per layer. We call the resulting assignment for all variables in $\mathcal{V}$ an *extended state* in contrast to a normal state that fixes only the values of the fluent variables.

$\mathcal{O}$ is a finite set of *durative actions*. An action $\langle C, E, \delta\rangle$ consists of a triple $C = \langle C_\vdash, C_\leftrightarrow, C_\dashv\rangle$ of partial variable assignments over $\mathcal{V}_l$ (called its *start*, *persistent*, and *end condition*, respectively), a tuple $E = \langle E_\vdash, E_\dashv\rangle$ of *start* and *end effects* and a duration variable $\delta \in \mathcal{V}_n$. $E_\vdash$ and $E_\dashv$ are finite sets of *conditional effects* $\langle c, e\rangle$. The *effect condition* $c = \langle c_\vdash, c_\leftrightarrow, c_\dashv\rangle$ is defined analogously to the operator condition C. Since in real world applications (and in the benchmark domains of the planning competition) conditional start effects do not depend on persistent or end conditions, we require for these effects that $c_\leftrightarrow = c_\dashv = \emptyset$. The *simple effect* e is either a logical effect of the form $v = w$ or a numeric effect of the form $v \circ v'$, where $v, v' \in \mathcal{V}_n$ and $\circ$ is one of the operators $+=$, $-=$, $*=$, $/=$, and $:=$.

We have made several requirements in this definition (e.g. $\mathcal{G}_n$ acyclic, $\mathcal{A}_l$ stratifiable). Notice, that all these are satisfied naturally when translating from temporal PDDL. The temporal numeric *SAS*$^+$ formalism captures all of PDDL 2.1 level 3 (Fox and Long 2003) except for duration inequalities and metrics (but could easily be extended by these features).

A *plan* for a temporal planning task is a collection of durative actions, each annotated with a start time point and a duration. It must be executable in the intuitive sense, respecting the action conditions and the so-called no-moving-targets restriction (Fox and Long 2003) that demands that there are no two actions that simultaneously make use of a value if one of the two is accessing the value to update it. Additionally, the state reached by the plan must satisfy the goal condition.

Search

To solve the planning tasks, i.e., to find plans, preferably of low makespan, we use a search method similar to that used in Sapa (Do and Kambhampati 2003). More precisely, we perform a heuristic search in the space of time-stamped states, where the two types of search steps are the insertion of a durative action at the current time point and the advancement of the current time by a certain increment.

A *time-stamped state* $\mathcal{S} = \langle t, s, E, C_\leftrightarrow, C_\dashv\rangle$ consists of a *time stamp* $t \geq 0$, an *extended state* s, a set E of *scheduled effects*, and two sets $C_\leftrightarrow$ and $C_\dashv$ of persistent and end conditions. A scheduled effect $\langle \Delta t, c_\leftrightarrow, c_\dashv, e\rangle$ consists of the remaining time $\Delta t \geq 0$ (until the instant when the effect triggers), persistent and end effect conditions $c_\leftrightarrow$ and $c_\dashv$ over $\mathcal{V}_l$, and a simple effect e. The conditions in $C_\leftrightarrow$ and $C_\dashv$ are annotated with time increments $\Delta t \geq 0$ and have to hold until instant $t + \Delta t$ (exclusively) for persistent conditions and at instant $t + \Delta t$ for end conditions.

We say that a condition holds in a time-stamped state iff it holds in the corresponding extended state. A time-stamped state $\mathcal{S}$ is *consistent* iff all unexpired persistent conditions and all end conditions whose due time has just been reached hold in $\mathcal{S}$. The application of a set of scheduled effects to an extended state s results in the extended state s' obtained from s by accordingly updating all fluents affected by a scheduled effect that triggers in s, and leaving the values of other fluents unaltered. Derived variables are evaluated in s' as usual after the fluents have been assigned their new values.

A *cleaned-up* time-stamped state is obtained from a consistent time-stamped state by applying all those scheduled effects whose due time has been reached to the extended state, and then dropping all scheduled effects and persistent conditions with reached due time. The *temporal progression* of a cleaned-up time-stamped state $\mathcal{S}$ is $\mathcal{S}$, if no more scheduled effects or conditions remain in $\mathcal{S}$. Otherwise, let Δt be the minimal time increment of all scheduled effects and conditions in $\mathcal{S}$. Then the temporal progression of $\mathcal{S}$ is like $\mathcal{S}$, except that the time-stamp is incremented by Δt, all time increments of scheduled conditions and effects are decremented by Δt, and scheduled effects with violated persistent conditions are removed.

The *repeated progression* of $\mathcal{S}$ is the sequence of time-stamped states that starts with $\mathcal{S}$ and is built up by repeatedly generating the temporal progression of the cleaned-up version of the preceding state in the sequence. A time-stamped state $\mathcal{S}$ can be *consistently progressed* if its repeated progression only contains consistent time-stamped states and if for any two successive states $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ in the repeated progression, their intermediate state that is like $\mathcal{S}_i$ except for having a time-stamp that is strictly between those of $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$, is consistent as well (this is needed to check persistent

conditions).

We can apply a durative action op in $\mathcal{S}$ by applying all start effects (including a subsequent axiom evaluation) and adding its end effects as scheduled effects and its persistent and end conditions to $C_{\leftrightarrow}$ and $C_{\dashv}$ respectively, using the operator duration as Δt. We say that an operator op is applicable in a time-stamped state $\mathcal{S}$ iff its start-condition holds in $\mathcal{S}$ and its application results in a time-stamped state $\mathcal{S}'$ that can be consistently progressed.

The search starts at time point 0 from the extended initial state and without any scheduled effects or conditions. The successors of a time-stamped state are all those time-stamped states that can be obtained by either inserting an applicable durative action at the current time point or by computing the temporal progression of the current state. In order to satisfy the no-moving-targets rule, small time increments of $\epsilon > 0$ are inserted after each action addition to a time-stamped state.

We perform an A* search, always expanding the time-stamped state $\mathcal{S} = \langle t, s, E, C_{\leftrightarrow}, C_{\dashv} \rangle$ from the open list that minimizes $f(\mathcal{S}) = t + h(\mathcal{S})$. Additionally, we use the deferred evaluation and preferred operator techniques known from the Fast Downward planning system (Helmert 2006). Since we use duplicate elimination, we never consider a state that differs from an already expanded state only in having a larger time stamp.

We return a plan as soon as a time-stamped state is reached that satisfies the goal and where no more scheduled conditions or effects remain. In our experiments, we use an any-time version, where the planner does not terminate upon finding a plan, but rather keeps searching for better plans and returns them in order of increasing quality.

Temporal (In)completeness

Note that our search space allows us to find plans for certain problems with required concurrency (Cushing et al. 2007). Consider, e.g., a problem where we want to mend a fuse (the goal is that `(mended f)` is true). This can only be achieved using the action `(mend-fuse f m)`, which requires the predicate `(light m)`, which is initially *false*, to be *true* across the whole execution of `(mend-fuse f m)` (including the end points of the action). The only action setting `(light m)` to *true* (with its start effect) is `(light-match m)`. However, the same action also has an end effect setting `(light m)` back to *false*. Therefore, in every plan for this problem, the action `(mend-fuse f)` must start after the start of the action `(light-match m)` and finish before the end of `(light-match m)`. A possible plan – as generated by our planner – is depicted in Fig. 4.

LPG-td and SGPlan cannot find a plan for this problem, since no sequential plan exists. Sapa also cannot solve the problem, since it does not consider any instant between the start and end point of `(light-match m)` for starting a new action. The only planners mentioned in this paper which are able to cope with such a problem are Crikey and TFD. We find a plan, since we add time increments of $\epsilon > 0$ after each action insertion.

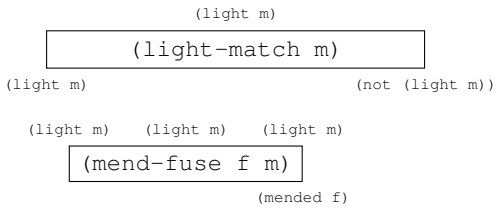


Figure 4: Plan for problem with required concurrency.

Context-enhanced Additive Heuristic

For guiding the search, we use a variant of the (inadmissible) context-enhanced additive heuristic extended to cope with numeric variables and durative actions. In order to explain our modifications, it is necessary to briefly introduce the "original" heuristic. We borrow the terminology and definitions from the original paper of Helmert and Geffner (Helmert and Geffner 2008). Hence, readers who are familiar with this work can safely skip this section.

The context-enhanced additive heuristic is defined for sequential *multi-valued planning tasks* which are tuples $\Pi = \langle V, s_0, s_\star, O \rangle$ where V is a set of logical (fluent) variables, s_0 is a state over V characterizing the initial situation, $s_\star$ is a partial state characterizing the goal situations, and O is a set of operators mapping one state into a possibly different state.

The difference to temporal numeric *SAS*$^+$ is the absence of axioms and numeric aspects and a different definition of operators: An operator in the sequential setting is a set of effects (or rules) of the form $v = w', z \rightarrow v = w$, where z is a partial state, v is a variable, and w and w' are values in $\mathcal{D}_v$. Such an effect means that if the current state s complies with z and s maps variable v to w', then the successor state s', resulting from the application of the operator, maps v to value w (while all mappings that are not changed by any effect of the operator stay the same). We sometimes write $o : v = w', z \rightarrow v = w$ to make clear that the rule is an effect of operator o.

Given a state s and an atom $v = w$, we use the notation $s[v = w]$ to denote 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 variable assignment to denote the state that is like s' for the variables in $\mathrm{dom}(s')$, and like s for all other variables.

The context-enhanced heuristic h^{cea} is defined as

$$h^{cea}(s) \stackrel{\text{def}}{=} \sum_{x \in s_\star} h^{cea}(x|x_s), \qquad (1)$$

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 rules $o : v = w', z \rightarrow v = w$, the condition $v = w'$ is achieved first, and the conditions in z are evaluated in the resulting state s. This leads to the

following equation:

$$h^{cea}(x|x') \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o:x'',z\to x} \big(1 + h^{cea}(x''|x') \\ \qquad + \sum\limits_{x_i \in z} h^{cea}(x_i|x_i'')\big) & \text{if } x \neq x' \end{cases} \tag{2}$$

In the nontrivial case, the first summand, 1, captures the cost of applying the minimizing operator o (assuming a unit-cost model), the second 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'. This state is denoted by $s(x''|x')$ and is obtained from

$$s(x''|x') \stackrel{\text{def}}{=} \begin{cases} s[x'] & \text{if } x'' = x' \\ s(x'''|x')[z'][x'', y_1, \ldots, y_n] & \text{if } x'' \neq x' \end{cases} \tag{3}$$

where x''' is the atom for which $o : x''', z' \to x''$ is the rule that yields the minimum in Eq. 2, and $y_1, \ldots, y_n$ are the heads of all rules that must trigger simultaneously with this rule (i.e., $o : x''', z_i' \to y_i$ for some $z_i' \subseteq z'$ for all $i = 1, \ldots, n$, for the same operator o). In other words, if $o : x''', z' \to x''$ is the best (cost-minimizing) achiever for atom x'' from x' according to Eq. 2, then before applying operator o atom x''' must be true. The state resulting from achieving x''' from x' is (recursively) obtained as $s(x'''|x')$. Since all conditions of $o : x''', z' \to x''$ must be true before o can be applied, we can update these values and obtain state $s(x'''|x')[z']$. After applying operator o, not only is atom x'' true but also the heads of all other rules that trigger simultaneously. Hence, the resulting state, also capturing all "side effects" of operator o, is $s(x'''|x')[z'][x'', y_1, \ldots, y_n]$.

Making the Heuristic Useful for Temporal and Numeric Planning

In order to use the context-enhanced additive heuristic for temporal and numeric planning, we need to answer two major questions:

1. How to transform durative actions into operators that are suitable for the heuristic computation?
2. How to deal with the numeric aspects of the planning task?

Our answer to the first question is the introduction of several types of so-called *instant actions*, which the next section presents in detail. The subsequent section explains the handling of the numeric aspects which is basically estimating the costs of changing the values of comparison variables.

Instant Actions

Since the context-enhanced additive heuristic is defined in terms of non-temporal operators, we emulate the temporal task by non-temporal *instant actions*. These newly introduced instant actions can be classified into several different groups.

The first group, which we call *compressed actions*, contains all instant actions that we derive by compressing a complete durative action. For this purpose, we use a transformation that is – at least similarly – also used by several other planning systems, e.g., by MIPS and LPG (Edelkamp 2003; Gerevini, Saetti, and Serina 2008): When compressing a condition triple, e.g. $\langle c_\vdash, c_\leftrightarrow, c_\dashv \rangle$, we remove all those persistent and end conditions that are made true by the associated operator op itself, i.e., all persistent and end conditions $v = w$ for which op contains a start effect $\langle c, v = w \rangle$ (ignoring the effect condition c). The triple is then compressed by building a single set of all remaining conditions. Note that by removing the distinction between start, persistent, and end conditions, the condition loses the property of being a (partial) variable mapping. Now, there can be atoms $v = w$ and $v = w'$, associated with the same variable. We transform a durative action by compressing all conditions (of actions and effects) and collecting all start and end effects except for those logical start effects that are overridden by an end effect. Notice that we keep all numeric start and end effects even if they share the affected variable.

In addition, we move all action conditions to the effect conditions and make sure that the affected variable of each logical effect also appears in a condition of the effect. If the condition c of a compressed logical effect $e = \langle c, v = w \rangle$ does not contain a condition on v, we introduce new effects $\langle c \cup \{v = w'\}, v = w \rangle$ for each $w' \in \mathcal{D}_v \setminus \{w\}$. These ideas are borrowed from Helmert and Geffner 2008 making it possible to write the effects in the same form they use. Using their notation, a compressed logical effect $\langle z \cup \{v = w'\}, v = w \rangle$ of instant action o is written as $o : v = w', z \to v = w$. Analogously, we write compressed numeric effects $\langle z, v \circ v' \rangle$ as $o : z \to v \circ v'$ ($\circ$ being among $+=$, $-=$, $*=$, $/=$, and $:=$). In addition, we assign each instant action o a cost $\mathrm{cost}(o)$ which is either a real number or a numeric variable. The cost of a compressed instant action o that is derived from a durative action $op = \langle C, E, \delta \rangle$ is simply the duration variable of that action, i.e., $\mathrm{cost}(o) = \delta$.

One problem with compressing a complete durative action is that we hide the state that is reached during the execution of the action (after applying the start effects) from the heuristic. For this reason, we introduce a second group of instant actions, called *start actions*, that cover only the start effects of a durative action. Nevertheless, we have to reflect the cost that results from applying the complete durative action. Hence, start actions include all conditions (start, persistent and end condition) that the compressed action also preserves, and the cost of a start action is the duration of the original durative operator. The only difference from the respective compressed action is that we do not add end effects.

Note that there is no reason to use analogous end actions. Whenever the heuristic calculation needs to use an end effect of a durative action, it can use the compressed action instead. An exception is the case where the associated durative action is already running in the evaluated time-stamped state $\mathcal{S}$. In this case we can consider the conditions of the durative action as already satisfied and estimate the cost of the transition as the remaining time of the running action. Thus, our third group of instant actions are the so-called *waiting*

actions: For each scheduled effect $\langle \Delta t, c_{\leftrightarrow}, c_{\dashv}, e \rangle$ in $\mathcal{S}$, we add a waiting action o with $\text{cost}(o) = \Delta t$ and effect $\rightarrow e$. If e is a logical effect $v = w$, we add conditions on $v = w'$ as above.

The last group of instant actions is derived from the logical *axioms*. Such an axiom $z \rightarrow v = w$ results in an instant action o with $\text{cost}(o) = 0$ and effects $\{v = w', z \rightarrow v = w \mid w' \in \mathcal{D}_v \setminus \{w\}\}$.

After answering the question of how to transform durative actions into instant actions, we still have to address the second question – how we can deal with the numeric aspects of the planning task.

Numeric Aspects

Eq. 2 shows that the heuristic estimate is based on the costs of making the atoms in the goal specification true, taking into consideration the costs that arise from making the necessary actions applicable. Since numeric variables do not directly occur in conditions or in the goal specification but only influence them via comparison variables, it is sufficient to estimate the cost of changing the values of these comparison variables.

Consider an unsatisfied condition $v = w$ with $v \in \mathcal{V}_c$ being a comparison variable and w being *true* or *false*, and let $v = v' \bowtie 0$ be the associated comparison axiom. The numeric variable v' represents an arithmetic expression over a set $\mathcal{F}(v)$ of numeric fluents. Our aim is to identify those instant actions that modify these fluents in such a way that $v = w$ becomes true, or which at least move v' closer to the desired value.

The set $\mathcal{F}(v)$ can easily be determined from the numeric dependency graph $\mathcal{G}_n$ by collecting the ancestors of v' with in-degree 0 (for example, in Fig. 3, $\mathcal{F}(aux_3) = \{w_1, n_1, h_1\}$). Based on this, we determine the set of rules (from the instant actions) that influence these variables, i.e., the set $\text{influencing}(v) = \{o : z \rightarrow v_1 \circ v_2 \mid v_1 \in \mathcal{F}(v)\}$. In the next step, we want to choose those rules from $\text{influencing}(v)$ that have a positive impact on the atom.

If we want to make an expression $v < 0$ or $v \leq 0$ true, we are interested in instant actions *decreasing* v, and if we want to make $v > 0$ or $v \geq 0$ true, we need to find instant actions *increasing* v. The converse holds if we want to make such an expression false. This leads to the following definition.

Let $w \in \{true, false\}$ be the current value of a comparison variable v whose value should be changed. We define

$$\bowtie^w = \begin{cases} < & \text{if } \bowtie \in \{<, \leq, =\} \text{ and } w = \mathit{false} \text{ or} \\ & \text{if } \bowtie \in \{>, \geq\} \text{ and } w = \mathit{true}, \\ > & \text{if } \bowtie \in \{>, \geq\} \text{ and } w = \mathit{false} \text{ or} \\ & \text{if } \bowtie \in \{<, \leq, =\} \text{ and } w = \mathit{true}. \end{cases} \quad (4)$$

A rule $o : z \rightarrow v_1 \circ v_2$ in $\text{influencing}(v)$ is *promising* to change value w of variable v in an extended state s (with $v = v' \bowtie 0$ being the associated comparison axiom), if

$$s[v_1 \circ v_2](v') \bowtie^w s(v') \quad \text{if } \bowtie \in \{<, \leq, \geq, >\}, \text{ or}$$
$$|s[v_1 \circ v_2](v')| \bowtie^w |s(v')| \quad \text{otherwise,}$$

where $s[v_1 \circ v_2]$ is the state that equals s except for the update $v_1 \circ v_2$ and a subsequent axiom evaluation. We refer to the set of promising rules as $\text{prom}(v = w, s)$.

Using this definition, we can extend Eq. 2 for costs and comparison variables as follows:[1]

$$h^{cea}(x|x') \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o: x'', z \rightarrow x} \big(c(s'') + h^{cea}(x''|x') + \sum\limits_{x_i \in z} h^{cea}(x_i|x_i'')\big) & \text{if } x \neq x', \\ & \text{var}(x) \notin \mathcal{V}_c \\ \min\limits_{o: z \rightarrow v \circ v' \in \text{prom}(x', s')} \big(c(s') + \sum\limits_{x_i \in z} h^{cea}(x_i|x_i')\big) & \text{if } x \neq x', \\ & \text{var}(x) \in \mathcal{V}_c \end{cases} \quad (5)$$

where $c(s) = \text{cost}(o)$ if $\text{cost}(o) \in \mathbb{R}$ and $c(s) = s(\text{cost}(o))$, otherwise. The states s' and s'' are the state corresponding to x', and the state after reaching x'' from x' (cf. Eq. 3)[2], respectively.

Then, the temporal variant of the heuristic is defined as

$$h^{cea}(\mathcal{S}) \stackrel{\text{def}}{=} \sum_{x \in s_\star} h^{cea}(x|x_s). \quad (6)$$

Note that, whereas the objective of the search is finding plans with a low parallel makespan, the heuristic does not take concurrency into account and merely sums over the durations of relevant actions, thereby effectively providing an estimate of the makespan of a sequential plan (or, identifying durations with costs, the total cost of a plan). We believe that these estimates are still informative, since typically there is a positive correlation between parallel and sequential makespans. The inadmissibility of the heuristic arises from different sources, specifically from the inherent sequentiality of the heuristic, from possible double counts of action durations in the form of start and compressed instant actions, and from the inadmissibility of the non-temporal context-enhanced additive heuristic.

To illustrate how the heuristic computation works, we will have another look at the gardening robots example introduced in Fig. 1.

Illustrating Example

Consider the simplified problem in which only f_2 needs to be watered (and f_1 and f_3 can be left alone) and there is only one robot r_1. Accordingly, the goal is to satisfy $h_2 - n_2 = 0$. We start with the comparison variable corresponding to this equality being *false*, and collect all those instant actions that modify either of the two compared fluents in such a way that we can hope to get h_2 and n_2 closer to each other. Since n_2 cannot be modified, a transition will be promising iff it modifies h_2 and moves it closer to n_2. There is only one such transition – specifically, the compressed action corresponding to the durative action `(water` r_1 ℓ_2 f_2`)`. The cost

[1] Since only logical variables can occur in conditions of instant actions, h^{cea} is only defined for atoms x with $\text{var}(x) \in \mathcal{V}_l$.

[2] Note that if the side effects y_i include updates of numeric fluents, the variables representing numeric terms containing these fluents are updated accordingly.

of this transition is the duration of the corresponding action, i.e., the difference $n_2 - h_2 = 50$ in the seed state. Besides the satisfied condition (in f_2 ℓ_2), the transition still has two unsatisfied conditions (at r_1 ℓ_2) and $w_1 \geq n_2 - h_2$. For the first one, we have to investigate the domain transition graph of the location variable of r_1. Clearly, we have to perform two successive walk transitions to get to ℓ_2: the (compressed) action (walk r_1 ℓ_1 ℓ_2) has a single unsatisfied precondition, (at r_1 ℓ_1), which can be achieved by (walk r_1 ℓ_0 ℓ_1). The latter action does not have any unsatisfied preconditions in the seed state. We can take the two transitions at costs of 40 and 70, respectively. Having completed the first precondition of the watering action, we still have to take care of the second one, i.e., $w_1 \geq n_2 - h_2$. The promising transitions here are those increasing h_2 or w_1. When we look for the cost-minimizing among these, it turns out to be cheapest to increase w_1 using the transition (refill r_1 ℓ_0) at a cost of 150. In total, this gives us an estimate of $150 + 70 + 40 + 50 = 310$, which in this case is actually a perfect estimate. While the underlying plan in this example is sequential, the heuristic computation would work analogously if estimating a concurrent plan.

Experimental Results

To evaluate the performance of our planner, we compared it to several state-of-the-art planning systems: SGPlan 6 (Hsu and Wah 2008) won the temporal satisficing track in IPC 2008. Since, surprisingly, a simple planning system – implemented by the IPC 2008 organizers to serve as a baseline – performed better than any of the competitors, we used it for evaluation as well. This system, in the following called BasePlanner, ignores all temporal aspects and solves the task sequentially using Metric-FF (Hoffmann 2003). The resulting plan is then rescheduled in a post-processing step. We mentioned that our search space is very similar to the one of Sapa (Do and Kambhampati 2003), so we also included this system into our comparisons. Crikey (Coles et al. 2009) features the ability to solve many temporally expressive planning tasks. The last planner we compared to is the successful LPG-td (Gerevini, Saetti, and Serina 2008) (we used the anytime variant of LPG by enabling its 'quality' mode). To implement the approach described in this paper, we extended the Fast Downward planning system (Helmert 2006).

The benchmark set we used for the evaluation consists of all deterministic temporal domains of IPC 2008. All experiments were conducted with a 2 GB memory limit and a 30 minutes time limit on a computer with a 2.66 GHz Intel Xeon CPU.

Table 1 shows the quality of the planning systems measured with the evaluation scheme of IPC 2008 that comprises both plan length and coverage: Suppose the shortest known plan for a task has makespan Q^*. Then each planner gets a score of Q^*/Q for each solved instance, where Q is the makespan of the resulting plan. The domain score is the sum of all the instance scores, assigning 0 to unsolved tasks. For some of the domains, the benchmark set contains different formulations (namely for elevators and openstacks). In these cases the total score of a planner includes only the score of the formulation on which it performs better.

The table shows that TFD performs better than all the other evaluated systems. Surprisingly, the older LPG-td still outperforms SGPlan 6 and BasePlanner, the official and unofficial winner of the last IPC. The other three planners are clearly behind.

We mentioned that the IPC score is based on plan quality as well as coverage. Observing that TFD solves comparatively few instances of the modeltrain, parcprinter and the transport domain, in the following we examine more closely where the strengths of our approach lie. For this purpose, we compare TFD to each of the other systems only on those tasks that are solved by both planners, thus focusing on the *quality* of the generated plans. The results are shown in Table 2.

The entries in this table state the average makespan ratios of the generated plans. A value greater than 1.0 indicates that the solutions found by TFD are better than the corresponding solutions found by the competing system. The last row is the sum of this ratios weighted by the number of compared plans per domain. The table shows that our approach clearly outperforms all other systems in terms of plan quality: the plans produced by the other systems are on average between 31% (LPG-td) and 60% (SGPlan 6) longer than those of TFD.

Discussion and Future Work

We have seen that the heuristic described in this paper works remarkably well on the IPC 2008 benchmark set.

Still, there is room for improvement: In addition to the problem relaxation used in the non-temporal variant of the context-enhanced additive heuristic, we relax the temporal features of planning problems as well. It appears to be promising to investigate whether some of these relaxations could be removed, leading to a more precise state evaluation, without increasing the computational effort too much.

One relaxation is not to distinguish between start and end conditions in instant actions. Recall that the original context-enhanced additive heuristic has the underlying assumption that first the condition on the affected variable is made true and only then the other conditions. It appears promising to adapt this assumption and to update the state after satisfying the start conditions before the costs for the end conditions are determined.

Another issue of the current version of the heuristic is its inherent sequentiality. We can take concurrency into account by either extracting relaxed sequential plans and rescheduling them using a Simple Temporal Network (STN) based approach, or by performing a greedy scheduling of subplans during the computation of the relaxed plan.

The last issue is that we exploit a certain property of the benchmark domains: actions that have a positive impact on a numeric resource usually are sufficient to already satisfy all conditions on this resource (e.g., it is not necessary to refuel twice in order to have enough fuel for a certain driving action). If this is not the case, our heuristic underestimates the cost of making a condition true, because it considers effects only qualitatively. This could be resolved by a more in-depth analysis of the actual quantitative changes.

Domain	BasePlanner	Crikey 3	LPG-td	Sapa	SGPlan 6	TFD
Crewplanning	16.19	22.59	12.76	—	22.44	28.72
Elevators	18.38	2.60	22.75	5.64	15.09	19.38
Modeltrain	11.92	—	—	—	11.11	0.96
Openstacks	18.14	20.67	14.35	25.90	12.49	26.66
Parcprinter	13.84	8.58	18.20	5.25	11.00	9.10
Pegsol	24.35	18.30	25.81	18.98	15.39	27.57
Sokoban	15.52	7.03	11.95	0.00	8.73	13.00
Transport	5.50	2.83	11.57	1.91	7.46	6.91
Woodworking	12.14	11.96	26.37	9.36	10.44	16.04
Overall	**135.97**	**94.55**	**143.76**	**67.02**	**114.15**	**148.34**

Table 1: Overall results using the evaluation scheme of IPC 2008. A dash indicates that the planner produced errors or gave wrong results for that domain.

Domain	BasePlanner	Crikey 3	LPG-td	Sapa	SGPlan 6
Crewplanning	(18) 1.14	(29) 1.38	(13) 1.01	—	(28) 1.39
Elevators	(17) 1.58	(10) 3.00	(23) 1.11	(12) 2.26	(17) 1.84
Modeltrain	(1) 1.05	—	—	—	(1) 1.01
Openstacks	(30) 1.55	(30) 1.33	(30) 2.66	(30) 1.04	(30) 2.44
Parcprinter	(13) 1.37	(13) 1.34	(12) 0.70	(5) 1.12	(13) 1.32
Pegsol	(28) 1.19	(28) 1.60	(28) 1.12	(24) 1.28	(18) 1.21
Sokoban	(13) 1.29	(9) 1.50	(12) 1.50	—	(9) 1.22
Transport	(7) 1.18	(6) 1.79	(7) 0.75	(3) 1.54	(10) 1.37
Woodworking	(27) 1.42	(27) 1.44	(28) 0.66	(20) 1.37	(21) 1.29
Overall	(154) **1.36**	(152) **1.55**	(153) **1.31**	(94) **1.35**	(147) **1.60**

Table 2: Pairwise plan quality comparisons to TFD on the instances that are solved by both approaches (their number is stated in parentheses). Scores greater than 1 indicate that TFD generates shorter plans.

One feature that distinguishes Crikey 3 and TFD from the BasePlanner, LPG-td, Sapa and SGPlan is their ability to cope – to a higher (Crikey 3) or lower (TFD) degree – with problems requiring plans with overlapping actions (see Figure 4), whereas the other planners only work on strictly temporally simple (Cushing et al. 2007) domains.

Conclusion

We presented an adaptation of the context-enhanced additive heuristic to temporal and numeric planning. Whereas the implemented search algorithm over the space of time-stamped states is similar to the algorithms used by other temporal planning systems, this particular heuristic has not been studied in the context of temporal planning before. We show that, as in the classical case, the heuristic often returns estimates that are sufficiently accurate to steer the search towards the goal rather fast. Even though we still have some ideas how the accuracy of the heuristic can be further improved (interaction of subproblems and concurrency of actions can be handled more accurately), the current version of TFD already outperforms all state-of-the-art temporal planning systems. It does not only find plans for a large number of problems, but in particular plans of higher quality compared to plans returned by other temporal planning systems.

Acknowledgments

This work was supported by the German Research Council (DFG) by DFG grant NE 623/10-2 and as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, http://www.avacs.org/), by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IME01-ALU (DESIRE), and by the EU as part of the Integrated Project CogX (FP7-ICT-2xo15181-CogX).

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *AIJ* 173(1):1–44.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proc. IJCAI 2007*, 1852–1859.

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

Edelkamp, S. 2003. Taming numbers and duration in the model checking integrated planning system. *JAIR* 20:195–238.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *AIJ* 172(8-9):899–944.

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

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AIJ* 173:503–535.

Hoffmann, J. 2003. The Metric-FF planning system: Translating ignoring delete lists to numeric state variables. *JAIR* 20:291–341.

Hsu, C.-W., and Wah, B. W. 2008. The SGPlan planning system in IPC-6. `http://manip.crhc.uiuc.edu/Wah/papers/C168/C168.pdf`.

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