Combining Static and Dynamic Models for Boosting Forward Planning

Cédric Pralet and Gérard Verfaillie

ONERA - The French Aerospace Lab, F-31055, Toulouse, France
{Cedric.Pralet,Gerard.Verfaillie}@onera.fr

Abstract. This paper presents an example of cooperation between AI planning techniques and Constraint Programming or Operations Research. More precisely, it presents a way of boosting forward planning using combinatorial optimization techniques. The idea consists in combining on one hand a *dynamic model* that represents the Markovian dynamics of the system considered (*i.e.* state transitions), and on the other hand a *static model* that describes the global properties that are required over state trajectories. The dynamic part is represented by so-called *constraint-based timed automata*, whereas the static part is represented by so-called *constraint-based observers*. The latter are modeled using standard combinatorial optimization frameworks, such as linear programming, constraint programming, scheduling, or boolean satisfiability. They can be called at any step of the forward search to cut it via inconsistency detection. Experiments show significant improvements on some benchmarks of the International Planning Competition.

1 Introduction

Combinatorial optimization frameworks such as Constraint Programming (CP [24,4]), Integer Linear Programming (ILP [23]), or Boolean Satisfiability (SAT [6]) can sometimes be used to efficiently model and solve planning and scheduling problems. This is the case with problems that involve a finite set of tasks and a finite set of resources with simple evolution schemes. However, when resource evolution models are more complex, when the state of the dynamic system at each step must be explicitly considered, or when the number of steps in plans is initially unknown, dynamic modeling frameworks, based on the notions of states, events, and transitions (automata or Petri nets [8], STRIPS [11], PDDL [12], or ANML [26]), are often more suitable.

Several strategies have been proposed in the last decades to gather the strengths of combinatorial optimization and dynamic modeling frameworks used for planning. See for example the use of constraints in the IxTeT planner [20], in the CAIP framework [14], in the CPT planner [28], or in the CNT framework [27]. See also the use of SAT/CP/ILP for solving planning problems unrolled over a fixed number of steps [17,5,7,29,18].

In this paper, we present a new generic combination scheme between combinatorial optimization and dynamic modeling, for planning and scheduling in

deterministic settings. The basic idea consists in using two kinds of models simultaneously: on one hand a *dynamic* state transition model, able to represent complex Markovian state and resource evolutions, and on the other hand a *static* combinatorial optimization model, able to express global properties that are required on state trajectories. The dynamic part is modeled in a generic framework called CTA for *Constraint-based Timed Automata* (Sect. 3). The static part is captured by so-called *constraint-based observers* (Sect. 4). In the context of forward planning (Sect. 5), the static part observes transitions made by the dynamic part and is used to detect inconsistent trajectories. Experiments (Sect. 6) show significant improvements on benchmarks from the International Planning Competition (IPC [16]). We first begin with an illustrative example (Sect. 2).

2 Illustrative Example

Problem description Let us consider a satellite whose mission is to take images of the Earth surface, to record them, and to download them to ground stations. To do this, it is equipped with an observation instrument, an on-board memory of size $Mmax$, and an antenna of rate Dr. In the following, $\mathcal{I}$ denotes the set of candidate images to be taken, and $\mathcal{W}$ the set of temporal windows during which a ground station is visible. Taking image $i \in \mathcal{I}$ starts at time St_i and ends at time Et_i. Storing it consumes memory M_i. Downloading it lasts time M_i/Dr and must be performed in one of the windows in $\mathcal{W}$. We impose that any observation performed must also be downloaded over the planning horizon. It is possible to take an image and to download another one in parallel. If image i is taken and downloaded, reward R_i is obtained. The goal is to maximize the sum of rewards over the planning horizon.

Static modeling This problem can be formalized as a scheduling problem involving a finite set of tasks. Basically, a task ta is defined by a start time $startOf(ta)$, an end time $endOf(ta)$, and a boolean variable $presenceOf(ta)$ representing its presence (tasks may be optional and therefore not present in schedules). It is possible to introduce, for each image $i \in \mathcal{I}$, three tasks ob_i, rc_i, and dl_i representing respectively the observation of i, the recording of resulting data, and its download. Physical constraints and user requirements are modeled by Constraints 1 to 9, in a language inspired from the scheduling features of the OPL language [15]. Constraint 1 enforces that if an observation is performed (resp. not performed), associated recording and download are performed too (resp. not performed). Constraint 2 sets start and end times of observations. Constraint 3 sets durations of downloads. Constraint 4 enforces that a download must be performed within one of the ground station visibility windows. Constraint 5 enforces that observation must precede download. Constraint 6 expresses that recording starts at the beginning of observation and ends at the end of download. Constraint 7 (resp. 8) enforces that observations (resp. downloads) cannot overlap. Last, Constraint 9 expresses that the piecewise constant function incremented by memory consumptions during recordings must always be less than the memory available on-board (*cumulative* constraint, using the *pulse* function of OPL).

Equation 10 gives the criterion to be optimized. The model obtained is static in the sense that the sets of tasks and constraints are initially completely defined.

$$\forall i \in \mathcal{I} : \mathit{presenceOf}(ob_i) = \mathit{presenceOf}(rc_i) = \mathit{presenceOf}(dl_i) \quad (1)$$
$$\forall i \in \mathcal{I} : (\mathit{startOf}(ob_i) = St_i) \wedge (\mathit{endOf}(ob_i) = Et_i) \quad (2)$$
$$\forall i \in \mathcal{I} : \mathit{endOf}(dl_i) - \mathit{startOf}(dl_i) = M_i/Dr \quad (3)$$
$$\forall i \in \mathcal{I} : \mathit{presenceOf}(dl_i) \rightarrow (\exists w \in \mathcal{W}, \mathit{contains}(w, dl_i)) \quad (4)$$
$$\forall i \in \mathcal{I} : \mathit{endBefore}(ob_i, dl_i) \quad (5)$$
$$\forall i \in \mathcal{I} : \mathit{span}(rc_i, \{ob_i, dl_i\}) \quad (6)$$
$$\mathit{noOverlap}(\{ob_i, i \in \mathcal{I}\}) \quad (7)$$
$$\mathit{noOverlap}(\{dl_i, i \in \mathcal{I}\}) \quad (8)$$
$$\textstyle\sum_{i \in \mathcal{I}} \mathit{pulse}(rc_i, M_i) \leq Mmax \quad (9)$$
$$\mathit{maximize} \textstyle\sum_{i \in \mathcal{I}} R_i \cdot \mathit{presenceOf}(ob_i) \quad (10)$$

Dynamic modeling Assume now that there is a maximum level of energy *Emax* available on-board, due to a maximum charge of batteries, that taking images consumes power *Pob* and downloading them consumes power *Pdl*, that the platform consumes constant power *Psat*, and that sun exposure produces power *Psun*. We assume that the evolution of the level of energy available follows a *piecewise linear* and *saturated* model (saturation due to the maximum level of energy that can be stored in batteries). Modeling such evolutions or more complex ones with standard scheduling frameworks can become quite difficult. To use a static model, a possible alternative is to consider the planning problem over a fixed number of steps. But such an approach does not scale well in practice, since the models obtained may contain a huge number of variables and constraints.

Instead, modelers can come back to *dynamic models*, based on the notions of states, events, time, and transitions. These models remain compact because they do not unroll the problem beforehand over a fixed horizon. More precisely, let *ob*, *dl*, and *sun* be boolean state variables representing respectively whether or not the satellite is currently observing, downloading, and exposed to sun. Let $mm \in [0..Mmax]$, $en \in [0, Emax]$, and t, be state variables representing respectively the current levels of memory and energy and the current time. For each visibility window $w \in \mathcal{W}$ of a ground station, let vis_w be a boolean state variable representing whether or not the satellite is in w. Finally, for each image $i \in \mathcal{I}$, let $dlable_i$ be a boolean state variable representing whether or not image i has been taken and can be downloaded.

It is then possible to define events representing the start or the end of an observation, download, station visibility, or sun exposure period. In a STRIPS-like language [11], the preconditions and effects of these events are detailed below (given a state variable x, $'x$ denotes the value of x at the previous step; if a state variable is not reassigned in an event effect, its value is assumed not to change). For instance, for any image $i \in \mathcal{I}$, event $startOb(i)$ can occur only if the

satellite is not observing. Immediately after the event, the satellite is observing and memory size M_i is reserved. This dynamic model captures Constraints 4 to 9 in the previous static model.

$$
\begin{array}{ll}
startOb(i \in \mathcal{I}): & endOb(i \in \mathcal{I}): \\
\quad \text{pre: } \neg' ob & \quad \text{pre: } 'ob \\
\quad \text{post: } ob, mm = 'mm - M_i & \quad \text{post: } \neg ob, dlable_i
\end{array} \tag{11}
$$

$$
\begin{array}{ll}
startDl(i \in \mathcal{I}, w \in \mathcal{W}): & endDl(i \in \mathcal{I}, w \in \mathcal{W}): \\
\quad \text{pre: } \neg' dl, 'vis_w, 'dlable_i & \quad \text{pre: } 'dl, 'vis_w \\
\quad \text{post: } dl, \neg dlable_i & \quad \text{post: } \neg dl, mm = 'mm + M_i
\end{array} \tag{12}
$$

$$
\begin{array}{ll}
startVis(w \in \mathcal{W}): & endVis(w \in \mathcal{W}): \\
\quad \text{pre: } \neg' vis_w & \quad \text{pre: } 'vis_w \\
\quad \text{post: } vis_w & \quad \text{post: } \neg vis_w
\end{array} \tag{13}
$$

$$
\begin{array}{ll}
startSun: & endSun: \\
\quad \text{pre: } \neg' sun & \quad \text{pre: } 'sun \\
\quad \text{post: } sun & \quad \text{post: } \neg sun
\end{array} \tag{14}
$$

The evolution of the level of energy with time between any two events, the first one occurring at time $'t$ and the second one at time t, is given by a piecewise linear saturated model:

$$en = \min(E_{max}, 'en + (t - 't) \cdot (P_{sun} \cdot 'sun - P_{ob} \cdot 'ob - P_{dl} \cdot 'dl - P_{sat})) \tag{15}$$

The model obtained is dynamic in the sense that it models the dynamic evolution of the system and that the set of events is not initially defined. It is Markovian in the sense that any transition only depends on the previous state and on the current event. It is deterministic in the sense that only one transition is possible starting from a given state and a given event.

Conclusion This example shows that, depending on the features to be considered, a modeler can define either a *static model* and constraints over the whole planning horizon, or a *dynamic model* involving states, events, and transitions. In this paper, we propose to combine the two kinds of models *i.e.*, to use Equations 1 to 15 simultaneously. The combination of models for boosting the resolution of a problem is not a new idea. It was already used in operations research, for example by combining constraint and linear programming. It was also already used for AI planning, e.g. in planner IxTeT [20]. The objective of this paper is to propose a new generic cooperation scheme, between on one hand a dynamic model, and on the other hand a static model based on combinatorial optimization. In order to combine these two types of models, a first option is to give priority to the static part and to use dynamic models as resource profile checkers. A second option is to give priority to the dynamic part and to use static models as global constraint checkers. Both options may be interesting. This paper explores the second one.

3 Constraint-based Timed Automaton (CTA)

We first introduce the modeling framework we use to express dynamic models.

State evolution model The attributes of the system considered are modeled by a finite set S of *state variables*[1]. Set S is assumed to contain one special variable denoted t representing the current time. A constraint-based timed automaton over S is then defined as the union of three elements: (1) an *initialization model* I which specifies how variables in S are initialized, (2) an *event transition model* T^E which defines the evolution of variables upon arrival of events, and (3) a *time transition model* T^Δ which defines how variables evolve with time between two successive events. Models I, T^E, and T^Δ are expressed as sets of constraints over appropriate sets of variables. All these elements are formally defined below.

Definition 1. *A* constraint-based event type *e over a set of state variables S is a triple $\langle Param_e, Pre_e, Post_e \rangle$ where:*

- *$Param_e$ is a finite set of variables called the event* parameters*;*
- *Pre_e is a finite set of constraints on $'S \cup Param_e$ called the event* preconditions *of e ($'S$ is a copy of S used to represent the previous state);*
- *$Post_e$ is a function $\mathbf{d}(S) \times \mathbf{d}(Param_e) \rightarrow \mathbf{d}(S)$ called the event* effects*; this function is defined by a finite set of constraints on $'S \cup Param_e \cup S$ (more precisely, as an acyclic set of equalities reassigning some variables in S).*

Example If variables *ob* and *mm* respectively represent the current observation state of the satellite and the current level of memory available on-board, triple $startOb = (\{i\}, \{\neg' ob\}, \{ob, mm = 'mm - M_i\})$ defines event type "start of observation". This event type involves one parameter ($Param_e = \{i\}$), one precondition ($Pre_e = \{\neg' ob\}$), and two effects ($Post_e = \{ob, mm = 'mm - M_i\}$).

In the following, given an event type e and an instantiation $p \in \mathbf{d}(Param_e)$ of its parameters, $e(p)$ denotes event type e instantiated by p. $e(p)$ is classically called a *ground event*. Events and associated state transitions are assumed to be instantaneous. We assume that, at a given step, one and only one event occurs, even if successive steps can have the same temporal position. The order in which events occur may thus influence the resulting state. Similar assumptions are made in classical timed automata [1].

Definition 2. *A* Constraint-based Timed Automaton *(CTA) is a tuple $\langle S, t, I, T^E, T^\Delta \rangle$ where:*

- *S is a finite set of* state variables*, containing one special variable t representing time;*
- *I, called the* initialization model*, is a set composed of one constraint $x = a$ per $x \in S$;*

[1] For any variable x, $\mathbf{d}(x)$ denotes its domain of values and, for any set X of variables, $\mathbf{d}(X)$ denotes the Cartesian product of the domains of values of the variables in X.

– T^E, *called the* event transition model, *is a set of constraint-based event types over* S;
– T^Δ, *called the* time transition model, *is a function* $\mathbf{d}(S) \times \mathbb{R}^+ \to \mathbf{d}(S)$, *defined by a finite set of constraints on* $'S \cup \{\delta\} \cup S$, *where* δ *is a variable representing the duration between the time step associated with* $'S$ *and the time step associated with* S; *we assume that this set of constraints contains one special constraint* $t = 't + \delta$, *and that the state does not change if* δ *takes value* 0.

Example To model the illustrative space example as a CTA, we can use $S = \{ob, dl, sun, mm, en, t\} \cup \{vis_w \mid w \in \mathcal{W}\} \cup \{dlable_i \mid i \in \mathcal{I}\}$. The initialization model is given by constraints such as $sun = 1$ or $en = 58$, depending on the initial state considered. Event transition model T^E is given by Equations 11 to 14, and time transition model T^Δ is given by Equation 15.

It must be emphasized that the definition of a CTA is close to the PDDL+ language [13] and to the timed automata framework [1]. With regard to these frameworks, it brings a unified constraint-based view.

The set of *valid* system trajectories induced by a CTA corresponds to the set of state sequences satisfying both the initialization and transition models:

Definition 3. *Let* $c = \langle S, t, I, T^E, T^\Delta \rangle$ *be a CTA. A* valid trajectory *for* c *is a sequence of states* $s_1 \to^{trans_1} s_2 \to^{trans_2} \dots s_n \to^{trans_n} s_{n+1}$ *where each transition* $trans_i$ *is either of the form "*(δ)*" (time transition of duration* δ*), or of the form "*$e(p)$*" (event transition due to ground event* $e(p)$*), and where:*

– *state* s_1 *satisfies constraints in* I;
– *for every* $i \in [1..n]$ *such that* $trans_i \equiv (\delta)$, *we have* $s_{i+1} = T^\Delta(s_i, \delta)$;
– *for every* $i \in [1..n]$ *such that* $trans_i \equiv e(p)$, *it holds that* $e \in T^E$, $p \in \mathbf{d}(Param_e)$, (s_i, p) *satisfies* Pre_e, *and* $s_{i+1} = Post_e(s_i, p)$.

Event arrival model CTAs model how the state of the dynamic system considered evolves, through event and time transitions. In another direction, it is possible to define the set of allowed event and time transitions. A first option used in STRIPS planning is to use no event arrival model (at each step, any feasible action is a candidate action). Another option used in PDDL consists in using the notion of durative action to link events of start and end of an action. In HTNs [22], the event arrival model is defined as a task network to be decomposed.

As this is not the main subject of the paper, we just indicate here that we use an event arrival model in which (1) some events are activated initially at some specific times (for example, *startSun* is activated at the start of each sun exposure period and $startVis(w)$ is activated at the start of visibility window w); (2) event transitions can activate new events in the future; for example, event $startOb(i)$ activates event $endOb(i)$ (if an observation starts, it must end), event $startDl(i, w)$ activates event $endDl(i, w)$ (if a download starts, it must end), and event $endOb(i)$ activates event $startDl(i, w)$ (if an observation ends, the image taken may be immediately downloaded); and (3) all events present in a trajectory must be activated at some step (either initially or by event transitions).

4 Constraint-based Observers

The previous section has described the state evolution model and the event arrival model used in CTAs. We now introduce the notion of *observers*, which will allow static and dynamic models to be linked in a common framework.

4.1 Definitions

Definition 4 introduces the notion of *observer*. Basically, an observer is a component which watches event transitions $('s, e(p), s)$ and time transitions $('s, \delta, s)$ of the dynamic system, and whose role is to detect inconsistency. To achieve this task, an observer maintains an internal state s_o that evolves with transitions observed. The evolution of s_o due to event (resp. time) transitions is given by a function denoted T_o^E (resp. T_o^Δ). An observer is equipped with a consistency function P_o associating value 0 with sequences $s_{o,1} \rightarrow \ldots \rightarrow s_{o,k}$ of observer states that are deemed inconsistent, and value 1 with the others. The *observer* terminology is inspired from automata and model-checking [10], where observers are used to determine whether or not all possible trajectories induced by a given automaton satisfy a specific property.

Definition 4. *Let $\langle S, t, I, T^E, T^\Delta \rangle$ be a CTA, and δ be a temporal duration variable. An* observer *is a tuple $\langle S_o, I_o, T_o^E, T_o^\Delta, P_o \rangle$ such that:*

- *S_o is a finite set of variables called the* observer state variables*;*
- *$I_o : \mathbf{d}(S) \rightarrow \mathbf{d}(S_o)$ is a function called the* observer state initialization function*;*
- *T_o^E is a function associating with each event type $e \in T^E$ a function $T_o^E(e) : \mathbf{d}(S) \times \mathbf{d}(Param_e) \times \mathbf{d}(S) \times \mathbf{d}(S_o) \rightarrow \mathbf{d}(S_o)$; T_o^E is called the* event transition observation function*; informally, $T_o^E(e)('s, p, s, 's_o)$ gives the observer state s_o obtained if an event transition $('s, e(p), s)$ is observed in state $'s_o$;*
- *$T_o^\Delta : \mathbf{d}(S) \times \mathbf{d}(\delta) \times \mathbf{d}(S) \times \mathbf{d}(S_o) \rightarrow \mathbf{d}(S_o)$ is a function called the* time transition observation function*; informally, $T_o^\Delta('s, \delta, s, 's_o)$ gives the observer state s_o obtained if a time transition $('s, \delta, s)$ is observed in state $'s_o$;*
- *$P_o : Trajs(S_o) \rightarrow \{0, 1\}$ is a function called the* consistency function *of the observer ($Trajs(S_o)$ denotes the set of partial observer state trajectories $s_{o,1} \rightarrow \ldots \rightarrow s_{o,k}$ over S_o); informally, P_o returns value 0 for partial trajectories that are deemed inconsistent, value 1 for the others.*

Example For readability reasons, observer state variables in S_o are represented by name in brackets, such as [x]. We consider the space example again and make the additional assumption that each image taken must start being downloaded to a ground station at most U units of time after the start of its realization. This property can be enforced using an observer $(S_o, I_o, T_o^E, T_o^\Delta, P_o)$ with:

- $S_o = \{[\mathsf{t}]\} \cup \{[\mathsf{obdone_i}] \mid i \in \mathcal{I}\} \cup \{[\mathsf{dldone_i}] \mid i \in \mathcal{I}\}$, where [t] represents a copy of the current time of the system and where, for every image $i \in \mathcal{I}$, variables [obdone$_i$] and [dldone$_i$] respectively memorize that the observation/download of image i has already been triggered in the past;

- $I_o = \{[\mathsf{t}] = 0\} \cup \{[\mathsf{obdone_i}] = 0 \mid i \in \mathcal{I}\} \cup \{[\mathsf{dldone_i}] = 0 \mid i \in \mathcal{I}\}$, expressing that initially, the time is null and no activity has begun;
- T_o^E an event transition observation function such that $T_o^E(startOb(i))$ changes the value of observer state variable $[\mathsf{obdone_i}]$ from 0 to 1, and $T_o^E(startDl(i))$ changes the value of $[\mathsf{dldone_i}]$ from 0 to 1;
- T_o^Δ a time transition observation function updating the current observer time by $[\mathsf{t}] = t$, with t the system state variable representing time;
- P_o the consistency function associating value 0 (inconsistency) with trajectories $s_{o,1} \to \ldots \to s_{o,k}$ such that there exists an image $i \in \mathcal{I}$ and two indices $j, j' \in [1..k]$ such that $j \leq j'$, $[\mathsf{obdone_i}]_j = 1$ (image i realized at step j), $[\mathsf{dldone_i}]_{j'} = 0$ (image i not downloaded yet at step j'), and $[\mathsf{t}]_{j'} - [\mathsf{t}]_j > U$ (maximum delay for download exceeded for image i).

Definition 4 may seem unusable in practice, since it involves a consistency function $P_o : \mathit{Trajs}(S_o) \to \{0, 1\}$ associating value 0/1 with each sequence of observer states. The number of such sequences being potentially huge or even infinite, it is not possible to define P_o in extension.

A possible way to solve this problem consists in expressing P_o implicitly. It is for example possible to consider that planners like TLPlan [3] or TALPlanner [19] use a consistency function P_o given by temporal logic reasoning. More precisely, TLPlan uses a set of temporal logic formulae representing properties to be satisfied by the system state trajectory induced by the solution plan. These formulae are checked at each step during search.

In this paper, we consider observers whose consistency function P_o is described implicitly not by temporal logic, but by constraint-based models. These observers involve, in addition to the observer state variables, a set of so-called *static variables*. The latter model global features of system state trajectories. Two kinds of constraints are imposed over static variables: *static constraints*, which correspond to standard constraints, and *dynamic constraints*, whose role is to establish connection between static variables and the observer state variables. These elements are formally defined below.

Definition 5. *A* constraint-based observer *is a tuple* $\langle S_o, I_o, T_o^E, T_o^\Delta, P_o, V_o, C_o^S, C_o^D \rangle$ *such that:*

- $\langle S_o, I_o, T_o^E, T_o^\Delta, P_o \rangle$ *is an* observer*;*
- V_o *is a finite set of variables called* observer static variables*;*
- C_o^S *is a set of constraints over* V_o *called the* observer static constraints*;*
- C_o^D *is a set of constraints over* $S_o \cup V_o$ *called the* observer dynamic constraints*;*
- consistency function P_o *applied to a partial trajectory* $ptraj : s_{o,1} \to \ldots \to s_{o,k}$ *associates value* 0/1 *with the* constraint satisfaction problem $(V_o, C_o^S \cup \{C_o^D(s_{o,i}) \mid i \in [1..k]\})$*, where for every* $i \in [1..k]$*,* $C_o^D(s_{o,i})$ *denotes the set of constraints over* V_o *obtained from* C_o^D *after having assigned variables in* S_o *to their value in* $s_{o,i}$*;* P_o *is said to be correct (or admissible) if and only if* $P_o(ptraj) = 0$ *implies that the problem is inconsistent;* P_o *is said to be complete if and only if* $P_o(ptraj) = 1$ *implies that the problem is consistent.*

The intuition behind Definition 5 is that a constraint-based observer accumulates constraints along trajectories and detects trajectory inconsistency when the set of static constraints in C_o^S, plus the accumulated dynamic constraints in C_o^D, is inconsistent. The reasoning performed by an observer may be incomplete *i.e.*, associating value 1 with inconsistent problems. This may be useful to obtain fast observers. It may be incorrect *i.e.*, associating value 0 with consistent problems. This may be useful to obtain more pruning at the price of possible loss of solutions.

Definition 5 allows the set of variables V_o and the set of constraints C_o to take the form of any standard combinatorial optimization problem: a mixed/integer linear problem (MIP), a boolean satisfiability problem (SAT), a pseudo-boolean optimization problem (PB), or a constraint satisfaction problem (CP). In case of a mixed/integer linear problem, consistency function P_o can be the consistency of the linear relaxation obtained using the simplex algorithm. In case of boolean satisfiability and pseudo-boolean optimization, P_o can result from unit resolution. As for constraint programming, P_o can result from any constraint propagation mechanism allowing inconsistent values to be removed from the domains of variables in V_o (P_o returns 0 when an empty domain is detected).

From a semantic point of view, constraint-based observers may express different kinds of knowledge: (a) they can enforce redundant knowledge with regard to the dynamic part and serve to prune search earlier; in particular, observers can model static relaxations of a dynamic planning problem; or (b) they can specify additional constraints whose expression in a dynamic Markovian model would be cumbersome. More generally, observers may offer a more global view or a different perspective on a planning problem. Combining perspectives and branching several observers to a CTA can be the key to reduce computing times. Elements such as *temporally extended goals* [2] can also be captured by observers. Last, it must be stressed that the definition of observers does not imply the addition of attributes to the initial CTA, as each observer maintains its own internal state. This way, observers can be plugged in and out easily. A global view of the framework considered is given in Fig. 1.

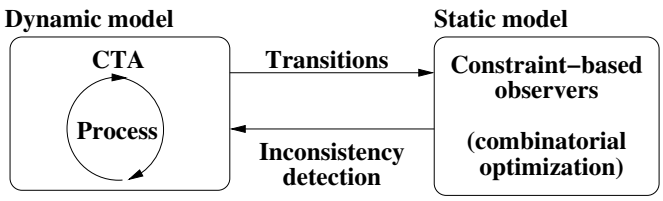

Fig. 1. Global view of the framework.

4.2 Example 1: a CP-observer for the illustrative space example

For the illustrative space example, we define an observer based on Constraint Programming (CP) and associated scheduling techniques.

Observer dynamic part $\langle S_o, I_o, T_o^E, T_o^\Delta \rangle$ Set S_o includes the following variables: variable [t] representing the current time, and, for each image $i \in \mathcal{I}$, four boolean state variables: $[\mathsf{obtriggered_i}]$ and $[\mathsf{dltriggered_i}]$ to represent respectively that the observation/download of image i has just been triggered, and $[\mathsf{obdone_i}]$ and $[\mathsf{dldone_i}]$ to memorize respectively that the observation/download of image i has already been triggered in the past.

Initially (I_o), $[\mathsf{t}] = 0$ and $[\mathsf{obtriggered_i}] = [\mathsf{obdone_i}] = [\mathsf{dltriggered_i}] = [\mathsf{dldone_i}] = 0$. Event transition function T_o^E expresses that $[\mathsf{obtriggered_i}]$ (resp. $[\mathsf{dltriggered_i}]$) takes value 1 if and only if event $startOb(i)$ (resp. $startDl(i)$) is observed. At every step, state variable $[\mathsf{obdone_i}]$ takes value $'[\mathsf{obdone_i}] \vee [\mathsf{obtriggered_i}]$, which means that an observation has been triggered in the past if either it was already triggered at the previous step, or it is triggered at the current step. Similarly, $[\mathsf{dldone_i}]$ is given by $'[\mathsf{dldone_i}] \vee [\mathsf{dltriggered_i}]$. Variable [t] is not modified by T_o^E, but when a time transition of duration δ occurs, T_o^Δ gives $[\mathsf{t}] = '[\mathsf{t}] + \delta$.

Observer static part $\langle V_o, C_o^S \rangle$ Observer static variables in V_o are, for each image $i \in \mathcal{I}$, tasks ob_i, dl_i, rc_i introduced in Sect. 2. Observer static constraints in C_o^S are Constraints 1 to 9, plus, to replace Equation 10, constraint $\sum_{i \in \mathcal{I}} R_i \cdot presenceOf(ob_i) > Best$, the best reward found so far.

Observer dynamic constraints C_o^D These constraints allow information to be transferred from the dynamic part to the static part of the observer. Observer dynamic constraints in C_o^D are Constraints 16 to 19. Constraint 16 expresses that if the observation of any image i is triggered, then task ob_i is present. Constraint 17 expresses that if any image i has not been already taken and if its observation start time St_i is outdated, then task ob_i is not present. Constraint 18 specifies that if the download of any image i is triggered, then task dl_i is present and its start time equals the current time. Last, Constraint 19 expresses that if any image i has not been downloaded yet, then either it is not downloaded, or its download starts after the current time.

$$[\mathsf{obtriggered_i}] \rightarrow presenceOf(ob_i) \tag{16}$$

$$(\neg[\mathsf{obdone_i}] \wedge ([\mathsf{t}] > St_i)) \rightarrow \neg presenceOf(ob_i) \tag{17}$$

$$[\mathsf{dltriggered_i}] \rightarrow (presenceOf(dl_i) \wedge (startOf(dl_i) = [\mathsf{t}])) \tag{18}$$

$$\neg[\mathsf{dldone_i}] \rightarrow (\neg presenceOf(dl_i) \vee (startOf(dl_i) \geq [\mathsf{t}])) \tag{19}$$

Observer consistency function P_o It can be the result of constraint propagation over the static variables, using efficient techniques to handle scheduling constraints (case of an incomplete, but correct, consistency function).

4.3 Example 2: an ILP-observer for the Trucks planning problem

We give a second example of observer, based on Integer Linear Programming.

Problem description The planning problem called *Trucks* is a benchmark of the International Planning Competition (IPC [16]). It involves a set $\mathcal{P}$ of packages, a set $\mathcal{L}$ of locations, and a set $\mathcal{T}$ of trucks. Each truck has an initial location ($Lit_{\tau,i} = 1$ if and only if i is the initial location of truck τ). Trucks can load/unload packages and drive between locations. They have a maximum capacity of Np packages. Driving from location $i \in \mathcal{L}$ to location $j \in \mathcal{L}$ takes time $Dd_{i,j}$. Each package $p \in \mathcal{P}$ must be transferred from its initial location Lip_p to its goal location Lgp_p. The goal is to minimize the makespan, defined as the latest delivery time of a package. Some complex dynamic aspects of the problem are modeled in the dynamic part, such as the fact that packages loaded in a truck must follow a LIFO scheme (Last In First Out), or the fact that trucks may wait and collaborate to deliver packages. The ILP-observer we define actually just computes a lower bound on the makespan to prune search. This bound is obtained by reasoning over a *static relaxation* of the problem.

Observer dynamic part $\langle S_o, I_o, T_o^E, T_o^\Delta \rangle$ We consider the following observer state variables:

- for each truck $\tau \in \mathcal{T}$ and each pair of distinct locations $i, j \in \mathcal{L}$, integer variable $[\mathsf{ntDone}_{\tau,\mathsf{i},\mathsf{j}}]$ representing the number of transfers from location i to location j already initiated by truck τ;
- for each package $p \in \mathcal{P}$, variable $[\mathsf{lnext}_\mathsf{p}] \in \mathcal{L}$ representing the next location of package p.

Observer static part $\langle V_o, C_o^S \rangle$ Observer static variables in V_o give a global view of the truck transfers to be performed over the whole planning horizon. More precisely, V_o includes the following variables:

- for each truck $\tau \in \mathcal{T}$ and each pair of distinct locations $i, j \in \mathcal{L}$, integer variable $nt_{\tau,i,j}$ representing the number of transfers of truck τ from i to j;
- for each truck $\tau \in \mathcal{T}$, for each location $i \in \mathcal{L}$, boolean variable $llt_{\tau,i}$ indicating whether or not the last location of τ is i.

Observer static constraints in C_o^S are Constraints 20 to 22. Constraint 20 expresses that each truck has a unique last location. Constraint 21 is a *flow conservation* constraint: for each location i, the number of transfers towards i, plus 1 if i is the initial location of τ, equals the number of transfers from i, plus 1 if i is the last location of τ. Constraint 22 states that the duration consumed by all transfers must be strictly lower than the best makespan *Best* found so far.

$$\forall \tau \in \mathcal{T} : \textstyle\sum_{i \in \mathcal{L}} llt_{\tau,i} = 1 \tag{20}$$

$$\forall \tau \in \mathcal{T}, \forall i \in \mathcal{L} : Lit_{\tau,i} + \textstyle\sum_{j \in \mathcal{L} \setminus \{i\}} nt_{\tau,j,i} = llt_{\tau,i} + \sum_{k \in \mathcal{L} \setminus \{i\}} nt_{\tau,i,k} \tag{21}$$

$$\forall \tau \in \mathcal{T} : \textstyle\sum_{i,j \in \mathcal{L}, i \neq j} nt_{\tau,i,j} \cdot Dd_{i,j} \leq Best - 1 \tag{22}$$

Observer dynamic constraints C_o^D Observer dynamic constraints in C_o^D are Constraints 23 and 24. Constraint 23 expresses that, for each truck τ, the number of transfers from i to j is greater than or equal to the number of transfers already initiated. Constraint 24 generates *LP-cuts* which boost linear problem solving, by using a bound on the total number of transfers from one set of locations $\mathcal{L}'$ to its complementary $\mathcal{L} \setminus \mathcal{L}'$. This bound is equal to number of transfers already initiated from $\mathcal{L}'$ to $\mathcal{L} \setminus \mathcal{L}'$, plus a bound on the remaining number of transfers, computed from the next positions of packages, from the package goal locations, and from the truck capacities.

$$\forall \tau \in \mathcal{T}, \forall i, j \in \mathcal{L}, i \neq j : \; nt_{\tau,i,j} \geq [\mathsf{ntDone}_{\tau,\mathsf{i},\mathsf{j}}] \tag{23}$$

$$\begin{aligned} \forall \mathcal{L}' \subset \mathcal{L} : \; & \textstyle\sum_{\tau \in \mathcal{T}, i \in \mathcal{L}', j \in \mathcal{L} \setminus \mathcal{L}'} nt_{\tau,i,j} \geq \\ & \textstyle\sum_{\tau \in \mathcal{T}, i \in \mathcal{L}', j \in \mathcal{L} \setminus \mathcal{L}'} [\mathsf{ntDone}_{\tau,\mathsf{i},\mathsf{j}}] + \\ & \lceil \#\{p \in \mathcal{P} \mid [\mathsf{lnext}_{\mathsf{p}}] \in \mathcal{L}', Lgp_p \in \mathcal{L} \setminus \mathcal{L}'\} / Np \rceil \end{aligned} \tag{24}$$

Observer consistency function P_o It can be the result of solving the linear relaxation of the problem (case of an incomplete, but correct, consistency function).

5 Forward Search Pruned by Observers

The algorithm we use is a depth-first search combining forward search on CTA and constraint-based reasoning on observers. To handle CTAs, the algorithm maintains a temporal database containing all future events and their arrival time. If current time is t and if the next time associated with an event in the database is $t' > t$, forward search first operates a time transition of duration $t' - t$. After that, one event e dated at t' is non deterministically chosen in the temporal database. An assignment p of $Param_e$ satisfying Pre_e is selected, and the effect of e is computed using $Post_e$. This event transition may activate future events, which are added to the temporal database. Each time an inconsistency is revealed, backtrack occurs and other choices are tried.

On top of this basic mechanism, constraint-based observers can prune search: whenever a new event/time transition is triggered, the observer computes its updated internal state s_o and adds constraints in $C_o^D(s_o)$ to its current set of constraints. Consistency function P_o is then applied in order to detect inconsistency. Ideally, P_o should both return 0 as often as possible, to prune search early, and be computed fast, since it must be applied at each node developed during the forward search. Possible trade-offs for the choice of P_o are the use of constraint propagation instead of full search for a CP-observer, the use of the simplex over the linear relaxation for an MIP-observer, or the use of unit resolution for a SAT or PB-observer.

Another key point in the quest for efficiency is the design of observers able to compute their consistency function P_o as incrementally as possible (not from scratch) when the current trajectory is extended, that is when new constraints are added. This point can be handled in various ways:

– CP-observers whose consistency function P_o is constraint propagation are already able to handle incremental constraint addition and re-propagation;
– for MIP-observers whose consistency function P_o is the consistency of the linear relaxation, each time the simplex is applied, the optimal real solution found Sol is recorded; when a new state s_o is reached and Sol satisfies all linear constraints in $C_o^D(s_o)$, then Sol is still valid; otherwise, the simplex must be called again; however, it can be "warmed-up" by using so-called *simplex bases* in LP, produced during previous applications of the simplex;
– for SAT or PB-observers, clauses learnt during search can be reused.

To improve search efficiency, all structures used (domains of values after propagation for CP-observers, simplex solutions and bases for MIP-observers, learnt clauses for SAT or PB-observers) should ideally be backtrackable. This means that combinatorial optimization tools used should either be already backtrackable, or give access to their internal data structures.

6 Experiments

The solver developed is written in Java and called *CoPlan*. Various kinds of observer can be used in *CoPlan*: (1) CP-observers based on *Choco* [9], (2) CP-observers based on *ILOG CPOptimizer* [15], (3) a dedicated CP-observer for pure constraint checking, (4) MIP-observers based on *LpSolve* [21], (5) MIP-observers based on *ILOG CPLEX* [15], and (6) SAT or PB-observers based on *SAT4J* [25]. All these observers are able to detect inconsistency.

Experiments were performed on an Intel i5-520 1.2GHz, 4GBRAM, and focused on optimal planning (planning with a criterion to be optimized). *CoPlan* was compared with the last version of CPT [28], the current best optimal planner for temporal domains. This planner takes as input PDDL models and performs a constraint-based search in the space of partial plans. Constraints used in CPT ensure the consistency of plans built with regard to the causal links existing between the establishment of preconditions and the application of actions. As *CoPlan* benefits from the additional knowledge contained in observers, *CoPlan* and CPT do not use the same models. The point here is just to determine how interesting it can be to combine static and dynamic models for solving a problem.

Results for IPC domains BlocksWorld (IPC2), Satellite (IPC3), and Trucks (IPC5) are given in Tables 1 to 3. For each instance, the maximum CPU time allowed is of 3600 seconds. Globally, we observed that adding observers can significantly boost complete forward search.

The BlocksWorld domain involves a set of blocks which are stacked on a table, in a certain initial configuration, and which must be successively unstacked and stacked in order to reach a given goal configuration. The objective is to minimize the number of steps necessary to reach the goal. The observer used contains a unique dynamic constraint: $[\mathsf{step}] + [\mathsf{nblocksNotValid}] < Best$, where $Best$, $[\mathsf{step}]$, and $[\mathsf{nblocksNotValid}]$ represent respectively the best number of steps found so far, the index of the current step, and the number of blocks

BLOCKSWORLD Instance	Opt	*CoPlan* CPU time	CPT CPU time
bw-10-0	34	0.13	0.04
bw-15-0	40	0.18	2.50
bw-20-0	60	0.28	-
bw-25-0	82	0.57	-
bw-30-0	94	141.93	-
bw-40-0	134	3.11	-
bw-50-0	170	7.10	-
bw-60-0	210	15.98	-

Table 1. *CoPlan* vs. CPT for BlocksWorld (IPC1); CPU times in seconds.

whose current position is not valid (the position of a block which must be on the table at the end is valid if and only if it is currently on the table; the position of a block b which must be on top of a block b' at the end is valid if and only if it is currently on top of b' and b' has a valid position). This constraint is correct because [nblocksNotValid] can be shown to be a lower bound on the number of remaining steps. Table 1 shows a superiority of *CoPlan* over CPT.

The Satellite domain involves a set of satellites taking images in given orientations. The objective is to minimize the makespan. For this domain, we built a CP-observer based on *ILOG CPOptimizer*. This observer uses the so-called *alternative* and *noOverlap* scheduling constraints available in that tool. We first performed experiments with an incorrect version of this observer, in which consistency function P_o returns 0 if and only if *CPOptimizer* does not find a solution to the current CP problem in less than 0.1 second. Table 2 shows that the best solution found with this incorrect observer, given in column (Ub), is almost always optimal. To prove optimality, we replaced the incorrect consistency function by a correct one, namely constraint propagation. With this correct observer, *CoPlan* is slightly faster than CPT on the largest instances. We believe that CPT is competitive thanks to its capacity to directly backtrack on the source of conflicts encountered during search. One advantage of *CoPlan* is that it would have no difficulty, in terms of modeling, to deal with other optimization criteria, whereas CPT can only optimize the makespan.

For the Trucks domain presented in Sect. 4.3, we gave for hard instances an initial upper bound on the makespan as an input to both *CoPlan* and CPT (see column (Ub) in Table 3). We used the ILP-observer presented in the same section, implemented using MIP solver *LpSolve*. The advantage of using an ILP-observer here is that ILP variables can have an unbounded domain, which serves to represent plans of unbounded length. Table 3 shows a superiority of *CoPlan* with regard to CPT. It also shows that the first lower bound on the makespan produced by the ILP-observer is often better than the first one produced by CPT, which is based on the generic planning heuristics H_2 (see columns firstLb). For instances 1 to 6, which involve a unique truck, the first lower bound produced by the ILP-observer is even equal to the optimal makespan.

SATELLITE Instance	Opt	*CoPlan* incorrect CPU time (Ub)	*CoPlan* correct CPU time	CPT CPU time
sat-05	105.2600	2.01 (105.2600)	0.74	0.06
sat-06	64.8245	1.45 (64.8245)	0.43	0.48
sat-07	60.2019	1.37 (60.2019)	0.61	0.15
sat-08	73.9540	15.50 (73.9540)	12.22	2.40
sat-09	81.0341	3.59 (81.0341)	19.08	1.35
sat-10	104.9550	2.78 (104.9550)	1.43	0.50
sat-11	115.8440	3.16 (115.8440)	0.77	1.05
sat-12	129.7527	54.55 (129.7527)	-	3162.81
sat-13	-	116.07 (113.4714)	-	-
sat-14	85.7729	25.77 (86.3763)	2975.42	3521.17
sat-15	80.0789	64.30 (80.0789)	2010.62	2212.29

Table 2. *CoPlan* vs. CPT for Satellite (IPC3, temporal version); CPU times in seconds.

TRUCKS		*CoPlan*		CPT	
Instance (Ub)	Opt	CPU time	firstLb	CPU time	firstLb
T-01 ($+\infty$)	843.2	0.27	843.2	0.01	815.6
T-02 ($+\infty$)	1711.4	0.14	1711.4	-	770.11
T-03 ($+\infty$)	1202.6	0.51	1202.6	0.14	901.5
T-04 ($+\infty$)	2629.4	0.43	2629.4	4.43	981.8
T-05 ($+\infty$)	1671.6	0.55	1671.6	-	617.0
T-06 ($+\infty$)	4319.0	1.10	4319.0	663.86	2048.82
T-07 (1500.0)	1236.6	491.1	1030.6	-	875.91
T-08 (2000.0)	1697.4	1839.5	1380.5	2454.56	1691.4
T-09 (2500.0)	2241.8	2160.2	2181.0	-	1746.8
T-10 (1800.0)	-	-	1372.6	-	1485.6
T-11 (2000.0)	-	-	1373.1	-	1314.6

Table 3. *CoPlan* vs. CPT for Trucks (IPC5, temporal version); CPU times in seconds.

7 Conclusion

This paper has presented a way of combining static and dynamic models for representing and solving planning and scheduling problems. The great flexibility of the framework proposed should allow the reasoning capabilities in both kinds of model to be efficiently combined for each problem at hand. For example, several observers could be combined for reasoning on the same problem. First results have been obtained with pruning observers which allow the search on dynamic models to be pruned early. In another direction, it would be interesting to explore the use of heuristic observers which would allow this search to be guided, by providing a heuristic choice between alternatives.

References

1. Alur, R., Dill, D.: A Theory of Timed Automata. Journal of Theoretical Computer Science 126(2), 183–235 (1994)
2. Bacchus, F., Kabanza, F.: Planning for Temporally Extended Goals. Annals of Mathematics and Artificial Intelligence 22(1-2), 5–27 (1998)
3. Bacchus, F., Kabanza, F.: Using Temporal Logics to Express Search Control Knowledge for Planning. Artificial Intelligence 16, 123–191 (2000)
4. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer Academic Publishers (2001)
5. van Beek, P., Chen, X.: CPlan: A Constraint Programming Approach to Planning. In: Proc. of the 16th National Conference on Artificial Intelligence (AAAI-99). pp. 585–590. Orlando, FL, USA (1999)
6. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press (2009)
7. Bockmayr, A., Dimopoulos, Y.: Integer Programs and Valid Inequalities for Planning Problems. In: Proc. of the 5th European Conference on Planning (ECP-99). pp. 239–251. Durham, UK (1999)
8. Cassandras, C., Lafortune, S.: Introduction to Discrete Event Systems. Springer (2008)
9. CHOCO Team: CHOCO, http://www.emn.fr/z-info/choco-solver/
10. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
11. Fikes, R., Nilsson, N.: STRIPS: a New Approach to the Application of Theorem Proving. Artificial Intelligence 2, 189–208 (1971)
12. Fox, M., Long, D.: PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)
13. Fox, M., Long, D.: Modelling Mixed Discrete-Continuous Domains for Planning. Journal of Artificial Intelligence Research 27, 235–297 (2006)
14. Frank, J., Jónsson, A.: Constraint-Based Attribute and Interval Planning. Constraints 8(4), 339–364 (2003)
15. IBM ILOG: Cplex optimization studio, http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/
16. IPC: International Planning Competition, http://ipc.icaps-conference.org/
17. Kautz, H., Selman, B.: Planning as Satisfiability. In: Proc. of the 10th European Conference on Artificial Intelligence (ECAI-92). pp. 359–363. Vienna, Austria (1992)
18. Kautz, H., Walser, J.: Integer Optimization Models of AI Planning Problems. Knowledge Engineering Review 15(1), 101–117 (2000)
19. Kvarnström, J., Doherty, P.: TALplanner: A Temporal Logic Based Forward Chaining Planner. Annals of Mathematics and Artificial Intelligence 30, 119–169 (2001)
20. Laborie, P., Ghallab, M.: Planning with Sharable Resource Constraints. In: Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95). pp. 1643–1651. Montréal, Canada (1995)
21. lp_solve Team: lp_solve, http://lpsolve.sourceforge.net/
22. Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research 20, 379–404 (2003)
23. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. John Wiley & Sons (1988)

24. Rossi, F., Beek, P.V., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)
25. SAT4J Team: SAT4J, http://www.sat4j.org/
26. Smith, D., Frank, J., Cushing, W.: The ANML Language. In: Proc. of the 18th International Conference on Automated Planning and Scheduling (ICAPS-08). Sydney, Australia (2008)
27. Verfaillie, G., Pralet, C., Lemaître, M.: How to Model Planning and Scheduling Problems using Timelines. The Knowledge Engineering Review 25(3), 319–336 (2010)
28. Vidal, V., Geffner, H.: Branching and pruning: An optimal temporal POCL planner based on constraint programming. Artificial Intelligence 170, 298–335 (2006)
29. Vossen, T., Ball, M., Lotem, A., Nau, D.: On the Use of Integer Programming Models in AI Planning. In: Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99). pp. 304–309. Stockholm, Sweden (1999)