Slice Encoding for Constraint-based Planning

Cédric Pralet and Gérard Verfaillie

ONERA, 2 avenue Édouard Belin, BP 74025, 31055 Toulouse Cedex 4, France
{Cedric.Pralet,Gerard.Verfaillie}@onera.fr

Abstract. In most of the constraint-based approaches to planning, the problem is unfolded over a given number of steps. Because this unfolded CSP encoding is very time and memory consuming, we propose on top of the CNT framework (*Constraint Networks on Timelines*) a more efficient slice CSP encoding which allows only a limited number of steps to be considered at each step of the search.

1 Introduction

In planning problems [1], we are looking for a plan that satisfies some properties and optimizes some criterion, but whose length *i.e.*, the number of steps it involves, is unknown and possibly unbounded. To overcome such a difficulty, following the seminal work of [2], constraint-based approaches to planning [3–6] model as a CSP the problem of finding a plan of fixed length H, with H incremented each time no plan of length H has been found. On the other hand, the CNT framework (*Constraint Network on Timelines* [7]) allows the length of the plan to be considered as a variable h, with a domain and with constraints linking it to any other variables. Following the lazy approach of [8], the problem can be unfolded only over the number of steps induced by the minimum value in the domain of h and extended only when this minimum value increases due to constraint propagation or branching choice.

However, what is common to all these approaches is an unfolded CSP encoding of the planning problem over a fixed or variable number of steps. Such an encoding may be very costly in terms of memory required to store all the data structures necessary to define and to manage the CSP and in terms of time required to create the CSP and to perform constraint propagation and search. For example, we observed some minutes or tens of minutes to create the CSP on some instances we wanted to solve. On some of them, there was even not enough memory to create the CSP without memory swaps. This led us to search for more compact CSP encodings of planning problems.

This paper presents the so-called *slice encoding* we developed on top of the CNT framework to allow only a limited number of steps, depending on the constraint graph structure, to be considered at each step of the search. The paper is organized as follows: first, the CNT framework is defined again *via* an illustrative example; second, the slice encoding is defined as well as a generic algorithm able to reason and to search on it; finally, experiments on planning benchmarks show its dramatic impact in terms of memory and time.

2 The CNT framework

2.1 An Illustrative Example

Let us consider the following planning problem which is a very simplified version of a real mission management problem for an Earth observation satellite.

We assume a satellite able to perform observations of specified areas of the Earth's surface. We consider a planning horizon $[Ts..Te]$ ($[a..b]$ denotes the set of integers between a and b). We assume N candidate observations on this horizon, numbered from 1 to N. Each observation $k \in [1..N]$ has a starting time $Ts[k]$ and an ending time $Te[k]$, with $Ts < Ts[k] < Te[k] < Te$. Two observations cannot overlap. To perform an observation, the instrument must be switched on at least Δon before starting observing. The instrument is initially off. Solar panels deliver a power Pe. When the instrument is on, a power $Ce > Pe$ is consumed. The initial energy level is Ei. Its minimum and maximum levels are $Emin$ and $Emax$. All data are assumed to be positive integers. The objective is to maximize the number of observations performed.

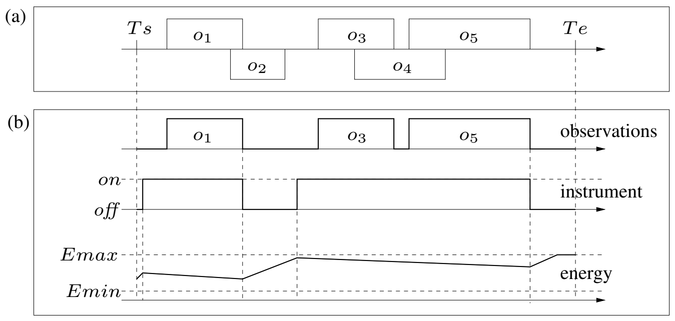

Fig. 1. Graphical representation of an instance (a) and of an optimal solution (b) of the mission management problem for an Earth observation satellite

Figure 1a represents an instance of this problem. Observations o_1 and o_2 are incompatible because of overlapping. Similarly, observation o_4 is incompatible with o_3 and o_5 because of overlapping. Figure 1b is a graphical representation of an optimal solution of this instance. Only observations o_1, o_3, and o_5 are performed. The instrument remains on between o_3 and o_5 because of insufficient time to switch it off. We can observe that the evolution of the energy level depends only on the status of the instrument. Let us use this planning problem to illustrate the basic definitions of the CNT framework.

2.2 Horizon Variables

Horizon variables are used to represent the number of steps to be considered.

Definition 1. *A horizon variable h is a variable whose domain of values is any subset of $\mathbb{N}$. We will use $\mathbf{D}(h)$ to denote the domain of a horizon variable h.*

In the planning problem described above, it may be useful to consider two horizon variables: a horizon variable ho to represent the number of steps in terms of observation and another one hi to represent the number of steps in terms of instrument status. If we associate a step with the start and the end of the horizon and with the start and the end of each observation performed, $\mathbf{D}(ho) = [2..2N+2]$: at least 2 steps when no observation is performed and at most $2N+2$ steps when all the candidate observations are performed. In the same way, if we associate a step with the start and the end of the horizon and with each instant at which the status of the instrument changes, $\mathbf{D}(hi) = \mathbf{D}(ho)$.

2.3 Time References and Timelines

Time references represent the temporal positions of the successive steps.

Definition 2. *A time reference t is a pair $\langle D, h\rangle$ where D is any subset of $\mathbb{R}$ and h a horizon variable. D is the domain of values of t and h is its horizon* i.e.*, the number of steps in t. We will use $\mathbf{D}(t)$ and $\mathbf{h}(t)$ to denote respectively the domain and the horizon of a time reference t.*

In our planning problem, we can consider two time references: one time reference to associated with observation, with $\mathbf{h}(to) = ho$ and $\mathbf{D}(to) = \{Ts, Te\} \cup (\cup_{k=1}^{N}\{Ts[k], Te[k]\})$, and one time reference ti associated with instrument status, with $\mathbf{h}(ti) = hi$ and $\mathbf{D}(ti) = \{Ts, Te\} \cup (\cup_{k=1}^{N}\ \{Ts[k] - \Delta on, Te[k]\})$ (it would be counterproductive to switch the instrument on strictly more than Δon before starting observing and to switch it off strictly after ending observing).

Timelines are used to represent the values of the relevant attributes of the system at the successive steps.

Definition 3. *A timeline x is a pair $\langle D, t\rangle$ where D is the domain of values of x and t its time reference. We will use $\mathbf{D}(x)$ and $\mathbf{t}(x)$ to denote respectively the domain and the time reference of a timeline x. For brevity, we will often use $\mathbf{h}(x)$ to denote the horizon of the time reference of a timeline x: $\mathbf{h}(x) = \mathbf{h}(\mathbf{t}(x))$.*

In our planning problem, we can consider three timelines: a timeline no to represent the number of the starting observation if any, a timeline in to represent the current instrument status, and a timeline en to represent the current level of energy, with $\mathbf{t}(no) = to$ and $\mathbf{t}(in) = \mathbf{t}(en) = ti$ (timelines in and en are fully synchronized). Moreover, we have $\mathbf{D}(no) = [0..N]$ (0 when no observation is starting), $\mathbf{D}(in) = \{0, 1\}$ (0 when the instrument is off), and $\mathbf{D}(en) = [Emin..Emax]$.

2.4 Static and Dynamic Variables

Definition 4. *A static variable is either a horizon variable, or any other variable independent from time references and timelines.*

In our planning problem, it may be convenient to associate with each candidate observation $k \in [1..N]$ a variable $st[k]$ of domain $[0..2N+2]$ to represent the observation step at which observation k is performed (0 when k is not performed). It may be also convenient to use a static variable obj of domain $[0..N]$ to represent the number of observations performed.

Dynamic variables are associated with steps of time references and timelines.

Definition 5. *A time reference t (resp. timeline x) and an assignment H of its horizon variable together induce a finite set of variables $\{t_i \mid i \in [1..H]\}$ (resp. $\{x_i \mid i \in [1..H]\}$). This set is empty when $H = 0$. All these variables share the same domain of values $\mathbf{D}(t)$ (resp. $\mathbf{D}(x)$).*

See Fig. 2 for a graphical representation of time reference ti and timeline en when $\mathbf{h}(ti) = 4$. For example, variable ti_3 represents the temporal position of step 3 and variable en_3 the energy level at step 3.

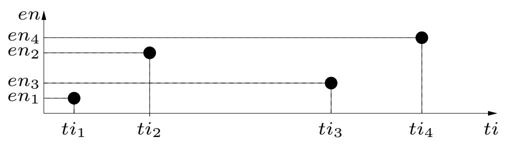

Fig. 2. Graphical representation of time reference ti and timeline en

2.5 Static and Dynamic Constraints

Static constraints are used to limit the possible assignments of static variables.

Definition 6. *A static constraint c is simply a CSP constraint [9] whose scope is limited to static variables.*

In our planning problem, the following static constraints can be defined:

$$obj = card\{k \in [1..N] \mid st[k] \neq 0\} \tag{1}$$

$$(Even(hi)) \wedge (Even(ho)) \wedge (hi \leq ho) \tag{2}$$

Constraint 1 defines the objective to be maximized as the number of observations performed. Constraint 2 limits the values of horizon variables hi and ho.

Dynamic constraints are used to limit the possible combinations of assignments of static and dynamic variables. The difficulty is that the set of dynamic variables associated with time references and timelines is not fixed. It depends on the assignments of the horizon variables. This leads us to a definition of a dynamic constraint which is not as obvious as the previous definitions are. We will use several examples to illustrate it.

Definition 7. *A dynamic constraint c is a tuple $\langle S, D, f \rangle$ where S is a finite set of static variables, D is a finite set of time references and timelines, and f is a function which associates a finite set of CSP constraints with each assignment H of the horizon variables of the timelines and time references in D. Variables in the scope of these induced CSP constraints must be either static variables in S or dynamic variables in the set of dynamic variables induced by H for time references and timelines in D.*

In our planning problem, the following dynamic constraints can be defined:

$$(to_1 = Ts) \wedge (no_1 = 0) \wedge (to_{ho} = Te) \wedge (no_{ho} = 0) \tag{3}$$
$$\forall i \in [2..ho-1],\ (no_i = 0) \leftrightarrow (no_{i-1} \neq 0) \tag{4}$$
$$\forall k \in [1..N], \forall i \in [2..ho],\ (no_{i-1} = k) \rightarrow ((to_{i-1} = Ts[k]) \wedge (to_i = Te[k])) \tag{5}$$
$$\forall k \in [1..N], \forall i \in [1..ho],\ (no_i = k) \leftrightarrow (st[k] = i) \tag{6}$$
$$(ti_1 = Ts) \wedge (in_1 = 0) \wedge (en_1 = Ei) \wedge (ti_{hi} = Te) \wedge (in_{hi} = 0) \tag{7}$$
$$\forall i \in [2..hi-1],\ in_i \neq in_{i-1} \tag{8}$$
$$\forall i \in [2..hi],\ en_i = \min(Emax, en_{i-1} + (ti_i - ti_{i-1}) \cdot (Pe - Ce \cdot in_{i-1})) \tag{9}$$
$$\forall i \in [2..ho],\ (no_{i-1} \neq 0) \rightarrow (DuringVal(in, 1, to_{i-1} - \Delta on, to_i)) \tag{10}$$

More precisely, Constraints 3 and 7 define the initial and final states. With regard to Definition 7, a constraint like $to_{ho} = Te$ is a dynamic constraint defined by the tuple $c = \langle \emptyset, \{to\}, f \rangle$ with f the function which associates with each assignment H of ho the unique unary CSP constraint $to_H = Te$ on dynamic variable to_H. Constraint 4 defines the alternation of observation starting and ending for timeline *no*. It corresponds to a dynamic constraint defined by the tuple $c = \langle \emptyset, \{no\}, f \rangle$ with f the function which associates with each assignment H of ho the set $\{(no_i = 0) \leftrightarrow (no_{i-1} \neq 0) \mid i \in [2..H-1]\}$ of $(H-2)$ binary CSP constraints, each one connecting dynamic variables no_i and no_{i-1}.

Constraint 5 links timeline *no* and time reference *to*. Together with the basic assumption of the CNT framework according to which temporal positions in any time reference are totally and strictly ordered (see Sect. 2.6 below), it guarantees that there is no overlapping between observations. Constraint 6 enforces consistency between timeline *no* and static variables $st[k]$. Constraint 8 defines the alternation of values 0 and 1 for timeline *in*. Constraint 9 describes the way the energy level evolves. Note that the minimum level of energy $Emin$ is guaranteed via the domain of timeline *en*. Constraint 10 uses a special constraint called *DuringVal* (not detailed here) and guarantees that the instrument is switched on at least Δon before starting observing and that it remains on until ending observing. Such a constraint is called a *synchronization constraint* because it links timeline *no* and timeline *in*, which do not share the same time reference.

2.6 Constraint Networks on Timelines (CNTs)

All these definitions can be put together in order to define *constraint networks on timelines.*

Definition 8. *A constraint network on timelines is a tuple* $N = \langle V, CS, T, X, CD\rangle$ *where* V *is a finite set of static variables,* CS *is a finite set of static constraints whose scopes are included in* V*,* T *is a finite set of time references whose horizons belong to* V*,* X *is a finite set of timelines whose time references belong to* T*, and* CD *is a finite set of dynamic constraints whose scopes in terms of static variables, time references, and timelines are respectively included in* V*,* T*, and* X*.*

It is assumed that a default dynamic constraint c_t *is associated with each time reference* $t \in T$*. This constraint enforces that the temporal variables associated with* t *are totally and strictly ordered:* $\forall t \in T, \forall i \in [1..\mathbf{h}(t) - 1]$, $t_i < t_{i+1}$.

The CNT which results from the modeling of our planning problem is defined by the tuple $\langle V, CS, T, X, CD\rangle$, with $V = \{ho, hi, obj\} \cup \{st[k] \mid k \in [1..N]\}$, $CS = \{c_i \mid i \in [1..2]\}$, $T = \{to, ti\}$, $X = \{no, in, en\}$, and $CD = \{c_i \mid i \in [3..10]\}$.

2.7 Assignments, Solutions, Consistency, and Optimality

Definition 9. *An assignment of a CNT* N *is an assignment of all its static variables (including all the horizon variables) and of all the induced dynamic variables. A solution is an assignment of* N *such that all its static constraints and all the CSP constraints induced by all its dynamic constraints are satisfied. A CNT is consistent if and only if it admits a solution. An objective variable* obj *is a static variable which is a function of other static or dynamic variables and whose domain is equipped with a total order* $\succ$*. A solution* Sol *is optimal iff there is no other solution* Sol' *such that* $Sol'[obj] \succ Sol[obj]$*.*[1]

In our planning problem, $obj = npo$ and $\succ = >$. Figure 3 represents an optimal solution of the CNT which results from the modeling of the instance of our planning problem described in Fig. 1a, with $Ts = 0$, $Te = 58$, $\Delta on = 3$, $Pe = 20$, $Ce = 22$, $Ei = 150$, $Emin = 50$, and $Emax = 300$. This is a tabular representation of the solution described in Fig. 1b. Assignments of static variables are omitted.

step	1	2	3	4	5	6	7	8
to	0	4	14	24	34	36	52	58
no	0	1	0	3	0	5	0	0
ti	0	1	14	21	52	58		
in	0	1	0	1	0	0		
en	150	170	144	284	222	300		

Fig. 3. Optimal solution of the CNT resulting from the modeling of the instance of the mission management problem described in Fig. 1a

[1] $A[v]$ denotes the value of variable v in assignment A.

2.8 Comparison with other modeling frameworks

From the constraint programming point of view, a CNT is a kind of dynamic (conditional) CSP [10] whose dynamic aspect comes from the assignment of the horizon variables and from the definition of the dynamic constraints. The CNT framework is in some way more specialized because it aims at modeling specific systems: discrete event dynamic systems. This justifies the presence of the basic notions of horizon and time, which do not appear in standard dynamic CSPs.

From the planning point of view, contrarily to classical frameworks such as PDDL [11], built around the notions of *action*, *precondition*, *effect*, *duration*, and resource *consumption*, the CNT framework is a more basic modeling framework, built around the notions of *time reference*, *timeline*, and *constraint*. This allows complex dynamic phenomena to be modeled. According to several features, the CNT framework is close to the CAIP framework used in the EUROPA planning system [12]. But the CNT framework is a pure CSP framework, which inherits the clear semantics and the flexibility of CSPs.

3 Slice CSP Encoding of Stationary Constraints

We can now present the slice CSP encoding of CNTs we propose. For the moment, this encoding is limited to CNTs where static constraints are of any kind, but dynamic ones are limited to so-called *stationary* dynamic constraints. Although such constraints already allow a wide range of planning problems to be modeled, extension to other dynamic constraints is discussed in Sect. 3.3.

3.1 Stationary dynamic constraints

To define stationary dynamic constraints, we first define *constraint generators.*

Definition 10. *A constraint generator c is a tuple $c = \langle S, G, R\rangle$ such that:*

- *S is a finite set of static variables;*
- *G is a finite sequence of pairs $\langle x, -k\rangle$ where x is either a time reference or a timeline and k a positive integer, and such that all time references and timelines x involved in G share the same horizon variable, denoted $\mathbf{h}(c)$;*
- *R is a subset of $\Pi_{v \in S}\mathbf{D}(v) \times \Pi_{(x,-k)\in G}\mathbf{D}(x)$, implicitly or explicitly defined.*

For each x involved in G, we will use $\mathbf{m}(c, x)$ to denote the memory of x in c, defined as $\mathbf{m}(c, x) = \max\{k \mid \langle x, -k\rangle \in G\}$. Finally, we will use $\mathbf{m}(c)$ to denote the memory of c, defined as $\mathbf{m}(c) = \max\{\mathbf{m}(c, x) \mid \langle x, .\rangle \in G\}$.

Definition 11. *Let $c = \langle S, G, R\rangle$ be a constraint generator. Given $i \in [\mathbf{m}(c) + 1..\max(\mathbf{D}(\mathbf{h}(c)))]$, the CSP constraint c_i generated by c at step i is a constraint whose scope is $S \cup \{x_{i-k} \mid \langle x, -k\rangle \in G\}$ and relation is R. The dynamic constraint generated by c is defined by the tuple $\langle S, D, f\rangle$ where $D = \{x \mid \langle x, .\rangle \in G\}$, and f is the function which associates with each assignment H of $\mathbf{h}(c)$ the set $\{c_i \mid i \in [\mathbf{m}(c) + 1..H]\}$ of CSP constraints* i.e.*, the dynamic constraint $\forall i \in [\mathbf{m}(c) + 1..\mathbf{h}(c)]$, c_i.*

Such a dynamic constraint is referred to as a *stationary* dynamic constraint because relation R does not depend on i. For example, let us consider the constraint generator $c = \langle \emptyset, \{\langle in, 0\rangle, \langle in, -1\rangle\}, R\rangle$, where in is the timeline introduced in Sect. 2 and R is the relation which includes all pairs (a, b) such that $a \neq b$. Intuitively, this constraint generator states that the value of timeline in at the current step must differ from the value of timeline in one step before. The memory of in in c is $\mathbf{m}(c, in) = 1$, which informally means that one step before the current step needs to be considered for in. In this case, the memory of c is $\mathbf{m}(c) = 1$ too. For each $i \in [\mathbf{m}(c) + 1..\max(\mathbf{D}(hi))]$, the CSP constraint c_i generated by c at step i is $in_i \neq in_{i-1}$ and the stationary dynamic constraint generated by c is $\forall i \in [2..hi]$, $in_i \neq in_{i-1}$.

When relation R depends on i, it is possible to remain stationary by introducing a timeline $\mathbf{st}(\mathbf{h}(c))$ whose value at each step i is equal to i. This implies introducing the stationary dynamic constraint $\forall i \in [2..\mathbf{h}(c)]$, $\mathbf{st}(\mathbf{h}(c))_i = \mathbf{st}(\mathbf{h}(c))_{i-1} + 1$ together with the special initialization constraint $\mathbf{st}(\mathbf{h}(c))_1 = 1$.[2] For example, constraint $ti_1 = Ts$ can be defined as the stationary dynamic constraint $\forall i \in [1..hi]$, $(\mathbf{st}(hi)_i = 1) \rightarrow (ti_i = Ts)$ and constraint $ti_{hi} = Te$ as the stationary dynamic constraint $\forall i \in [1..hi]$, $(\mathbf{st}(hi)_i = hi) \rightarrow (ti_i = Te)$.

In the following, given the bijection existing between a constraint generator and its associated stationary dynamic constraint, constraint generators are directly considered as stationary dynamic constraints.

3.2 Slice Encoding of Stationary CNTs

It is now possible to define stationary CNTs and their slice encoding:

Definition 12. *A stationary CNT $N = \langle V, CS, T, X, CD\rangle$ is a CNT such that all the dynamic constraints in CD are stationary.*

For each time reference or timeline $x \in T \cup X$, $\mathbf{m}(x)$ denotes the memory of x, defined as $\mathbf{m}(x) = \max\{\mathbf{m}(c, x) \mid c \in CD\}$.

Definition 13. *Let $N = \langle V, CS, T, X, CD\rangle$ be a stationary CNT. The slice encoding of N is the CSP $\langle \mathbf{slV}(N), \mathbf{slC}(N)\rangle$ such that:*

- $\mathbf{slV}(N)$ *is the set of* slice variables *generated by N, defined as $V \cup \{x^{(-k)} \mid (x \in T \cup X) \wedge (k \in [0..\mathbf{m}(x)])\}$ with $\mathbf{D}(x^{(-k)}) = \mathbf{D}(x)$ for each k ($x^{(-k)}$ is just a variable name);*
- $\mathbf{slC}(N)$ *is the set of* slice constraints *generated by N, defined as $CS \cup \{c^{(0)} \mid c \in CD\}$ where, for each constraint generator $c = \langle S, G, R\rangle \in CD$, $c^{(0)}$, called the slice constraint associated with c, is the CSP constraint whose scope is $S \cup \{x^{(-k)} \mid \langle x, -k\rangle \in G\}$ and relation is R.*

Given a stationary CNT, its slice encoding can be built automatically. For example, let us consider the part of our illustrative planning problem limited

[2] This special initialization constraint, which is not stationary, will be handled specifically by the algorithms defined in Sect. 4 (see function *sliceSolve*).

to horizon variable hi, time reference ti, timelines in and en, and Constraints 7 to 9. To get a slice encoding, a timeline $\mathbf{st}(hi)$ is introduced to represent the current step and Constraints 7 to 9 are transformed into the following stationary dynamic constraints (associated constraint generators are omitted):

$$\forall i \in [2..hi],\ \mathbf{st}(hi)_i = \mathbf{st}(hi)_{i-1} + 1 \tag{11}$$

$$\forall i \in [1..hi],\ (\mathbf{st}(hi)_i = 1) \rightarrow ((ti_i = Ts) \wedge (in_i = 0) \wedge (en_i = Ei)) \tag{12}$$

$$\forall i \in [1..hi],\ (\mathbf{st}(hi)_i = hi) \rightarrow ((ti_i = Te) \wedge (in_i = 0)) \tag{13}$$

$$\forall i \in [1..hi],\ (2 \leq \mathbf{st}(hi)_i \leq hi - 1) \rightarrow (in_i \neq in_{i-1}) \tag{14}$$

$$\begin{aligned}&\forall i \in [1..hi],\ (\mathbf{st}(hi)_i \geq 2) \rightarrow \\ &\qquad (en_i = \min(Emax, en_{i-1} + (ti_i - ti_{i-1}) \cdot (Pe - Ce \cdot in_{i-1})))\end{aligned} \tag{15}$$

In this particular case, all the time references and timelines have a memory of 1. But, in general, time references and timelines can have any memory greater than or equal to 0. According to Def. 13, the resulting slice encoding is as follows:

$$\mathbf{st}(hi)^{(0)} = \mathbf{st}(hi)^{(-1)} + 1 \tag{16}$$

$$(\mathbf{st}(hi)^{(0)} = 1) \rightarrow ((ti^{(0)} = Ts) \wedge (in^{(0)} = 0) \wedge (en^{(0)} = Ei)) \tag{17}$$

$$(\mathbf{st}(hi)^{(0)} = hi) \rightarrow ((ti^{(0)} = Te) \wedge (in^{(0)} = 0)) \tag{18}$$

$$(2 \leq \mathbf{st}(hi)^{(0)} \leq hi - 1) \rightarrow (in^{(0)} \neq in^{(-1)}) \tag{19}$$

$$\begin{aligned}&(\mathbf{st}(hi)^{(0)} \geq 2) \rightarrow \\ &\qquad (en^{(0)} = \min(Emax, en^{(-1)} + (ti^{(0)} - ti^{(-1)}) \cdot (Pe - Ce \cdot in^{(-1)})))\end{aligned} \tag{20}$$

Informally speaking, the slice encoding of a CNT contains all the constraints that must be checked at each step with regard to previous steps. Contrarily to an unfolded encoding whose size is in the best case a linear function of the number of steps to be considered, the size of a slice encoding is constant, only a function of the time reference and timeline memories. In this case, it involves only 8 variables and 5 constraints.

$Step$	1	2	3	4	5	6	7	8	9	10
in	in_1	in_2	in_3	in_4	in_5	in_6	in_7	in_8	in_9	in_{10}
en	en_1	en_2	en_3	en_4	en_5	en_6	en_7	en_8	en_9	en_{10}
ti	ti_1	ti_2	ti_3	ti_4	ti_5	ti_6	ti_7	ti_8	ti_9	ti_{10}

Unfolded CSP encoding

$\mathbf{st}(hi)$	$\mathbf{st}(hi)^{(-1)}$	$\mathbf{st}(hi)^{(0)}$
in	$in^{(-1)}$	$in^{(0)}$
en	$en^{(-1)}$	$en^{(0)}$
ti	$ti^{(-1)}$	$ti^{(0)}$

Slice CSP encoding

Fig. 4. Comparison between a unfolded and a slice CSP encoding when $hi = 10$

3.3 Extension to Other Constraints

The slice encoding presented in Sect. 3.2 works only on stationary dynamic constraints, that is on constraints that link dynamic variables that share the same horizon variable. For example, Constraint 10 in our illustrative planning problem, which links time reference *to*, timeline *no*, and timeline *in* in order to synchronize observations and instrument status, cannot be managed.

Fortunately, if each horizon is used by at most one time reference, it is possible to produce an automatic slice encoding (not detailed here) of so-called *synchronization* dynamic constraints, which use functions linking timelines having distinct time references, like $DuringVal$. When an horizon h is shared by two references t, t', it is always possible to create an additional variable h', to take $\mathbf{h}(t') = h'$ instead of $\mathbf{h}(t') = h$, and to add constraint $h = h'$, so that h is used by only one time reference. As a result, it is possible to build a complete slice encoding of our illustrative planning problem and of many similar problems.

However, some constraints, such as $allDifferent(x_i \mid \ i \in [1..\mathbf{h}(x)])$, remain uncovered. To deal with them, several options could be considered: to exhibit an automatic slice encoding as done with synchronization dynamic constraints, to reformulate the constraint by introducing additional variables in order to get a stationary dynamic constraint, or to come back to an unfolded encoding for x.

4 Reasoning and Searching on a Slice CSP Encoding

Functions 1, 2, 3, and 4 show the pseudo-code of an algorithm able to reason and to search on a slice encoding of any stationary CNT.

Algorithm 1: Main function

```
1  $N = \langle V, CS, T, X, CD \rangle$ a stationary CNT, with $H$ the set of horizon variables
2  sliceSolve($N$)
3  begin
4      $slV \leftarrow V \cup \{x^{(-k)} \mid (x \in T \cup X) \wedge (k \in [0..\mathbf{m}(x)])\}$
5      $slC \leftarrow CS \cup \{c^{(0)} \mid c \in CD\}$
6      $A \leftarrow \{\}$
7      foreach $h \in H$ do $\mathbf{D}(st(h)^{(0)}) \leftarrow \{1\}$
8      $(slV, slC, A) \leftarrow$ propagateAndShift($N, slV, slC, A$)
9      return recSolve($N, slV, slC, A$)
10 end
```

The main function *sliceSolve* automatically creates the slice encoding from the definition of the static and dynamic constraints, launches the system by setting to 1 the value at slice 0 of the step variable associated with each horizon variable, then calls functions *propagateAndShift* and *recSolve*.

Function *recSolve* is a standard depth-first tree search algorithm. The only restriction is that variables on which to branch must be either static variables

Algorithm 2: Recursive search function

```
1  $\mathbf{recSolve}(N, slV, slC, A)$
2  begin
3      $nasV \leftarrow \{v \in V \mid (|\mathbf{D}(v)| > 1)\}$
4      $nadV \leftarrow \{x^{(0)} \mid (x \in T \cup X) \wedge (|\mathbf{D}(x^{(0)})| > 1) \wedge (\mathbf{st}(\mathbf{h}(x))^{(0)} \leq \min(\mathbf{D}(\mathbf{h}(x))))\}$
5      $naV \leftarrow nasV \cup nadV$
6      if $(naV = \varnothing)$ then
7          return $A.\{(v, a) \mid v \in V, a \in \mathbf{D}(v)\}$
8      else
9          $(A'', opt) \leftarrow (null, +\infty)$
10         choose $v \in naV$ and a partition $\{d_1, d_2\}$ of $\mathbf{D}(v)$
11         for $i = 1$ to $2$ do
12             $\mathbf{D}(v) \leftarrow d_i$
13             $(slV', slC', A') \leftarrow \mathbf{propagateAndShift}(N, slV, slC \cup \{obj < opt\}, A)$
14             if $(\forall v' \in slV', |\mathbf{D}(v')| > 0)$ then $A' \leftarrow \mathbf{recSolve}(N, slV', slC', A')$
15             if $(A' \neq null)$ then $(A'', opt) \leftarrow (A', A'[obj])$
16         return $A''$
17 end
```

or dynamic variables at slice 0 when the minimum value in the domain of the associated horizon variable makes this slice mandatory. This allows the search to be performed systematically forward.

Algorithm 3: Propagate and shift function

```
1  $\mathbf{propagateAndShift}(N, slV, slC, A)$
2  begin
3      $(slV, slC) \leftarrow \mathbf{propagate}(slV, slC)$
4      $shift \leftarrow true$
5      while $((\forall v \in slV, |\mathbf{D}(v)| > 0) \wedge shift)$ do
6          $shift \leftarrow false$
7          $shH \leftarrow \{h \in H \mid (\forall x \in X \cup T \mid \mathbf{h}(x) = h,\ |\mathbf{D}(x^{(0)})| = 1) \wedge (\mathbf{st}(h)^{(0)} \leq$
           $\min(\mathbf{D}(h)))\}$
8          if $(shH \neq \varnothing)$ then
9              foreach $h \in shH$ do $(slV, slC, A) \leftarrow \mathbf{shift}(N, h, slV, slC, A)$
               $(slV, slC) \leftarrow \mathbf{propagate}(slV, slC)$
10             $shift \leftarrow true$
11     return $(slV, slC, A)$
12 end
```

Function *propagateAndShift* alternates classical propagations and shifts. Shifts occur on a horizon variable h each time all the associated dynamic variables at slice 0 are assigned and the minimum value in the domain of h makes a shift mandatory (the horizon is not reached yet).

Algorithm 4: Shift function

```
shift($N, h, slV, slC, A$)
begin
    foreach $c \in CD \mid h(c) = h$ do
        $slC \leftarrow slC \cup \{\mathbf{project}(c^{(0)}, V)\}$
        reinit($c$)
    foreach $x \in X \cup T \mid \mathbf{h}(x) = h$ do
        $A \leftarrow A.\{(x_{\mathbf{st}(h)^{(0)}}, x^{(0)})\}$
        for $k = \mathbf{m}(x)$ to 1 do $\mathbf{D}(x^{(-k)}) \leftarrow \mathbf{D}(x^{(-k+1)})$
        $\mathbf{D}(x^{(0)}) \leftarrow \mathbf{D}(x)$
    return $(slV, slC, A)$
end
```

As for function *shift*, it handles changes to be performed on the slice encoding when shifting. First, each dynamic constraint is projected to take into account the assignment of dynamic variables before forgetting it. For example, let us assume a constraint $x^{(0)} \leq v$ where x is a timeline and v a static variable. Let us assume that x must be shifted and that $x^{(0)} = 3$. Constraint $3 \leq v$ must be posted before shifting the slice encoding. Second, dynamic variable assignments at slice 0 are recorded and shifting operations are performed. Note that, due to the shifting operations, the domain of a dynamic variable $x^{(-k)}$ may decrease or increase. This contrasts with classical CSP solving where variable domains can only decrease along a search branch.

To make things more concrete, we show in Fig. 5 an example of execution of the algorithm previously described on a very small instance of the slice encoding defined by Constraints 16 to 20 in Sect. 3.2. We assume the following data: $Ts = 0$, $Te = 8$, $N = 1$, $Ts[1] = 5$, $Te[1] = 6$, $\Delta on = 3$, $Pe = 0.5$, $Ce = 1.5$, $Ei = 4$, $Emin = 2$, and $Emax = 10$. In fact, there is only one candidate observation which requires the instrument to be on at least over the interval [2..6]. Figures 5a and 5b show the slice encoding and the initialization. Figures 5c to 5e show an alternation of propagations and shifts. Figure 5f shows a branching choice followed by another alternation of propagations and shifts (Figs. 5g to 5i) which ends with inconsistency detection (there is not enough energy to switch on the instrument). After a backtrack (Fig. 5j) followed by an alternation of propagations and choices (Figs. 5k to 5m), a solution is found (the instrument is not switched on and the candidate observation is not performed).

It can be proved that, if all domains are finite (including the domains of the horizon variables), then this algorithm terminates and returns an optimal solution when the CNT is consistent and no solution when it is inconsistent. The key to the proof is the fact that, thanks to the forward approach, for each stationary dynamic constraint of the form $\forall i \in [\mathbf{m}(c) + 1..\mathbf{h}(c)]$, c_i , the algorithm guarantees at each step the satisfaction of the constraint $\forall i \in [\mathbf{m}(c)+1..\mathbf{st}(\mathbf{h}(c))^{(0)}]$, c_i.

	(-1)	(0)
st	[0..4]	[0..4]
in	0, 1	0, 1
en	[2..10]	[2..10]
ti	0, 2, 6, 8	0, 2, 6, 8
h : [2..4]		

Slice encoding
(a)

	(-1)	(0)
st	[0..4]	1
in	0, 1	0, 1
en	[2..10]	[2..10]
ti	0, 2, 6, 8	0, 2, 6, 8
h : [2..4]		

Initialization
(b)

	(-1)	(0)
st	0	1
in	0, 1	0
en	[2..10]	4
ti	0, 2, 6, 8	0
h : [2..4]		

Propagation
(c)

	(-1)	(0)
st	1	[0..4]
in	0	0, 1
en	4	[2..10]
ti	0	0, 2, 6, 8
h : [2..4]		

Shift
(d)

	(-1)	(0)
st	1	2
in	0	0, 1
en	4	[5..8]
ti	0	2, 6, 8
h : [2..4]		

Propagation
(e)

	(-1)	(0)
st	1	2
in	0	0, 1
en	4	[5..8]
ti	0	2
h : [2..4]		

Choice $ti^{(0)}=2$
(f)

	(-1)	(0)
st	1	2
in	0	1
en	4	5
ti	0	2
h : 4		

Propagation
(g)

	(-1)	(0)
st	2	[0..4]
in	1	0, 1
en	5	[2..10]
ti	2	0, 2, 6, 8
h : 4		

Shift
(h)

	(-1)	(0)
st	2	3
in	1	0, 1
en	5	∅
ti	2	6, 8
h : 4		

Propagation
Inconsistency
(i)

	(-1)	(0)
st	1	2
in	0	0, 1
en	4	[5..8]
ti	0	6, 8
h : [2..4]		

Backtrack to e
Post $ti^{(0)} \neq 2$
(j)

	(-1)	(0)
st	1	2
in	0	0, 1
en	4	[7..8]
ti	0	6, 8
h : [2..4]		

Propagation
(k)

	(-1)	(0)
st	1	2
in	0	0, 1
en	4	[7..8]
ti	0	8
h : [2..4]		

Choice
$ti^{(0)} = 8$
(l)

	(-1)	(0)
st	1	2
in	0	0
en	4	8
ti	0	8
h : 2		

Propagation
→Solution
(m)

Fig. 5. An example of execution of the algorithm (*st* stands for **st**(*hi*))

5 Experiments

The algorithm defined in Sect. 4 has been implemented in a CNT solver called SCOT (*Solver for Constraints On Timelines*) developed on top of the *Choco2* constraint solver (http://choco-solver.net). The implementation includes the management of synchronization dynamic constraints. As SCOT can handle unfolded and slice encodings of a same CNT, we are able to compare both approaches. Experiments were run on a SunUltra45, 1.6GHz, 1GBRAM.

Figure 6 shows the memory and time consumed by CSP creation on three planning problems of the International Planning Competition. The time limit was of 30 minutes. Unsurprisingly, results show the dramatic impact of a slice encoding. On the third problem, which involves synchronization constraints, an unfolded encoding cannot be produced within the time limit except for two instances, whereas a sliced encoding can be built for all instances.

Figure 7 shows the anytime quality profiles, i.e. the way plan quality evolves with cpu-time, on three instances of the second problem (*satellite-time*). On

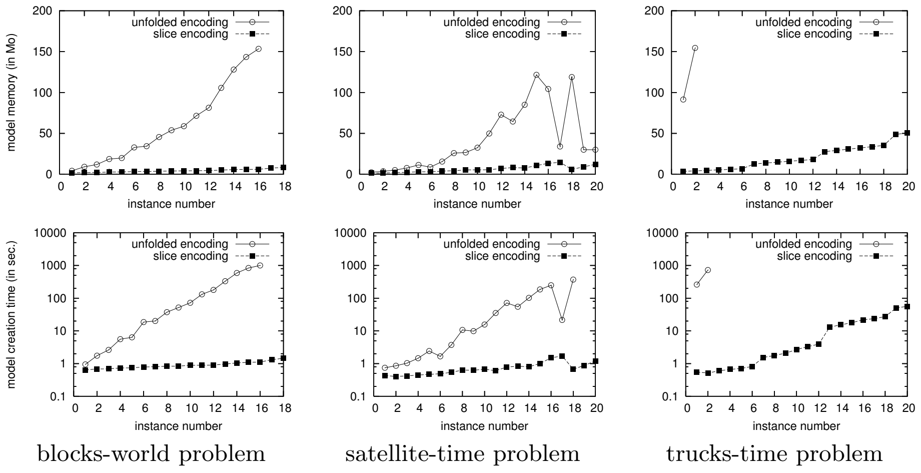

Fig. 6. CSP creation (first row: memory; second row: time)

these instances, plan quality corresponds to the makespan, to be minimized. The profiles include the time necessary to create the problem. Search algorithm parameters used are the same for both encodings. Again, results show the positive impact of the slice encoding. On the three instances, the algorithm working on a slice encoding is already producing good quality solutions when the same algorithm working on an unfolded encoding is still creating the CSP. However, the search with a slice encoding can start faster than the search with an unfolded encoding but be overtaken by the latter one. This is due to contradictory effects of the slice encoding: it generates smaller problems, but it limits look-ahead propagation and can result finally in a slower search. To make up for this negative effect, slice encodings with limited look-ahead could be considered.

The comparison of SCOT with other planners is out of the scope of this paper and can be found in [13].

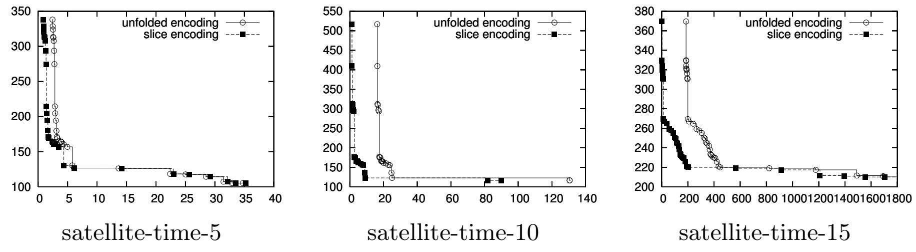

Fig. 7. Evolution of the makespan (y-axis) with cpu-time in seconds (x-axis)

6 Conclusion

This paper shows (1) that the slice encoding is an alternative to the usual unfolded CSP encoding of planning problems, (2) that it is equivalent to the unfolded one thanks to a forward search approach and to changes in standard CSP algorithms, and (3) that it is far more efficient in terms of memory and time required by CSP creation and in terms of anytime quality profile. Finally, we would like to acknowledge the Choco2 developers for fruitful interactions, especially Hadrien Cambazard and Charles Prud'homme.

References

1. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
2. Kautz, H., Selman, B.: Planning as Satisfiability. In: Proc. of the 10th European Conference on Artificial Intelligence (ECAI-92), Vienna, Austria (1992) 359–363
3. van Beek, P., Chen, X.: CPlan: A Constraint Programming Approach to Planning. In: Proc. of the 16th National Conference on Artificial Intelligence (AAAI-99), Orlando, FL, USA (1999) 585–590
4. Do, M., Kambhampati, S.: Planning as Constraint Satisfaction: Solving the Planning-Graph by Compiling it into CSP. Artificial Intelligence **132**(2) (2001) 151–182
5. Miguel, I., Shen, Q., Jarvis, P.: Efficient Flexible Planning via Dynamic Flexible Constraint Satisfaction. Engineering Applications of Artificial Intelligence **14**(3) (2001) 301–327
6. Lopez, A., Bacchus, F.: Generalizing GraphPlan by Formulating Planning as a CSP. In: Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico (2003) 954–960
7. Verfaillie, G., Pralet, C., Lemaître, M.: Constraint-based Modeling of Discrete Event Dynamic Systems. Journal of Intelligent Manufacturing (2008) Published online.
8. Pralet, C., Verfaillie, G.: Using Constraint Networks on Timelines to Model and Solve Planning and Scheduling Problems. In: Proc. of the 18th International Conference on Automated Planning and Scheduling (ICAPS-08), Sydney, Australia (2008) 272–279
9. Rossi, R., van Beek, P., Walsh, T., eds.: Handbook of Constraint Programming. Elsevier (2006)
10. Mittal, S., Falkenhainer, B.: Dynamic Constraint Satisfaction Problems. In: Proc. of the 8th National Conference on Artificial Intelligence (AAAI-90), Boston, MA, USA (1990) 25–32
11. Fox, M., Long, D.: PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research **20** (2003) 61–124
12. Frank, J., Jónsson, A.: Constraint-Based Attribute and Interval Planning. Constraints **8**(4) (2003) 339–364
13. Pralet, C., Verfaillie, G.: Forward Constraint-based Algorithms for Anytime Planning. In: Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09), Thessaloniki, Greece (2009)