Forward Constraint-based Algorithms for Anytime Planning

Cédric Pralet and **Gérard Verfaillie**
ONERA, 2 av. Edouard Belin, 31400 Toulouse, France
{Cedric.Pralet,Gerard.Verfaillie}@onera.fr

Abstract

This paper presents a generic anytime forward-search constraint-based algorithm for solving planning problems expressed in the CNT framework (*Constraint Network on Timelines*). It is generic because it allows many kinds of search to be covered, from complete tree search to greedy search. It is anytime because some parameter settings, together with domain-specific knowledge, allow high quality plans to be produced very quickly and to be further improved. It is forward because it systematically considers the decisions to be made in a chronological order. It is finally constraint-based because it is built on top of the CNT framework which is an extension of the CSP framework able to model discrete event dynamic systems and because it is implemented on top of the *Choco* constraint programming tool from which it inherits all the constraint handling machinery. Experimental comparisons are made in terms of quality profile with other domain-dependent and domain-independent planners.

Introduction

During the last decades, several approaches were proposed in the field of domain-dependent planning (Ghallab, Nau, and Traverso 2004) and were shown to induce dramatic gains in computation time on several domains. Some of these approaches, such as HTNs (*Hierarchical Task Networks*) used for example in the SHOP2 planner (Nau et al. 2003), express domain-dependent knowledge *via* task decomposition methods which help to structure the search and to avoid the exploration of useless combinations of primitive tasks. Other approaches express this knowledge *via* control rules which are checked at each step of the search. In the TLplan planner (Bacchus and Kabanza 2000), these rules take the form of first order LTL formula (*Linear Temporal Logic*). In TALplanner (Kvarnström and Doherty 2001), they are expressed in the TAL formalism (*Temporal Action Logics*).

Recently, another way of performing domain-dependent planning has been proposed in a modeling framework called CNT for *Constraint Network on Timelines* (Verfaillie, Pralet, and Lemaître 2008). The CNT framework is an extension of the CSP framework (*Constraint Satisfaction Problems* (Rossi, Beek, and Walsh 2006)) introduced to model discrete event dynamic systems and the properties one knows or one wants to verify or to enforce on them. Its modeling approach is close to the one of the *Constraint-based Attribute and Interval Planning* framework (Frank and Jónsson 2003), used in the domain-dependent EUROPA planner. However, the CNT framework differs in the basic modeling elements it uses: horizon variables to represent the number of steps to consider, time references to represent the temporal positions of each step, and timelines to represent the values of system attributes at each step.

The first algorithms that were defined in the CNT framework (Pralet and Verfaillie 2008) are optimal algorithms, able to produce plans that are guaranteed to be optimal with regard to a given criterion. Thanks to the additional knowledge expressed *via* constraints, these algorithms were shown to induce significant gains in computation time when compared with existing domain-independent optimal planning approaches. However, when compared with other domain-dependent planning approaches, they are not always able to produce quickly high quality plans. In other words, their anytime quality profile *i.e.*, the way plan quality evolves with computation time, is not very good, even though this anytime profile is crucial in many situations for which one wants a high quality plan to be produced by some strict deadline or as fast as possible (Zilberstein 1996).

This paper tackles the design of an anytime planning algorithm in the CNT framework. This algorithm is based on a forward search scheme which considers the decisions to be made in a chronological order. It is developed on top of a restarted depth-first search CSP solving algorithm, extended to solve CNTs. It is generic, but contains several parameters that can be easily set depending on the particular planning problem to be solved. Parameters allow for example a large scope of search procedures to be covered, from a complete tree search to a greedy stochastic one. They allow also any variable and value heuristics to be specified. This open tunable algorithm is the core of SCOT, our *Solver for Constraints On Timelines*, implemented on top of the *Choco 2.0* free constraint programming tool[1].

The paper is organized as follows: first, the CNT framework is defined again *via* an illustrative example; then we present the generic anytime forward-search constraint-based

[1] Choco: http://choco-solver.net/.

algorithm used in SCOT together with its possible parameter settings; last, experimental results are provided in order to show that the proposed approach produces plans whose quality is quickly better than the one of plans produced by other domain-independent or domain-dependent planners.

The CNT Framework

An illustrative example

Let us consider a planning problem for a team of unmanned air vehicles (UAVs). This problem involves a set of vehicles numbered from 1 to Nv and a set A of areas numbered from 1 to Na. The set A is partitioned into a set As of areas that should be *scanned* in order to localize an object and a set Ai of areas of which a *image* should be taken. For memory limitation reasons, each vehicle v cannot take more than $Ni(v)$ images. There is also a unique takeoff and landing area Ho, numbered 0, for all the vehicles. All the vehicles are initially landed and must be landed at the end of the mission. Each vehicle can take off at most once. With each pair a, a' of areas, is associated the duration $Du(a, a')$ needed to fly from a to a' and to perform the action required for a' (to scan, to take image, or to land when $a' = 0$, with the takeoff duration included when $a = 0$). The team is given the following mission: within a given maximum duration Md, maximize the number of visited areas and, as a secondary criterion, minimize the total duration of the mission. Figure 1 shows an instance of this problem involving 2 vehicles and 7 areas.

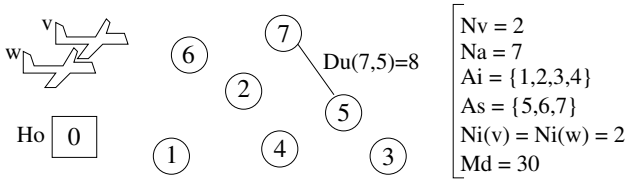

Figure 1: Instance of the planning problem for a team of UAVs.

Let us use this planning problem to illustrate the basic definitions of the CNT framework.

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 is possible to associate one horizon variable h_v with each vehicle v. This variable represents the number of steps in the activity plan of v. The minimum number of steps is equal to 1 (when v is not used and does not take off). The maximum number is equal to $Na + 1$ (when v visits all the areas and then goes home; visiting the same area several times is physically possible, but counterproductive). So, $\forall v \in [1..Nv], \mathbf{D}(h_v) = [1..Na + 1]$.

Time references

Time references are used to 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 associate two time references with each vehicle v: one time reference ts_v to represent the start time of each step in the activity plan of v and one time reference te_v to represent the end time of each step. Both share the same horizon variable h_v. The minimum time to be considered is 0 and the maximum is the maximum mission duration Md. So, $\forall v \in [1..Nv]$, $\mathbf{h}(ts_v) = \mathbf{h}(te_v) = h_v$ and $\mathbf{D}(ts_v) = \mathbf{D}(te_v) = [0..Md]$.

Timelines

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. To be short, 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, the relevant attributes at each step are the visited area and the duration of this visit. So, we associate two timelines with each vehicle v: one timeline at_v to represent the area and one timeline du_v to represent the duration. Both timelines share the same time reference ts_v and are consequently fully synchronized. However, between vehicles, timelines are not synchronized. Informally, time reference ts_v (resp. te_v) represents the start (resp. end) time of the visit of area at_v, including the travel from at_{v-1} to at_v and the visit of at_v itself. Note that time reference te_v has no associated timeline. We have: $\forall v \in [1..Nv]$, $\mathbf{t}(at_v) = \mathbf{t}(du_v) = ts_v$, $\mathbf{D}(at_v) = [0..Na]$, and $\mathbf{D}(du_v) = [0..Dumax]$, with $Dumax = max_{a,a' \in [0..Na]} Du(a, a')$.

Static 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 is convenient to associate with each area a a static variable vv_a of domain $[0..Nv]$ to represent the vehicle in charge of a (0 when a is not visited). It is also useful to associate with each vehicle v a static variable na_v of domain $[0..Na]$ to represent the number of areas visited by v. Last, we introduce a static variable e of domain $[0..Md]$ to represent the mission duration and a static variable o to represent the optimization criterion.

Dynamic variables

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 dynamic time variables $\{t_i \,|\, i \in [1..H]\}$ (resp. dynamic timeline variables $\{x_i \,|\, i \in [1..H]\}$). This set is empty when $H = 0$. All these variables share the same domain of values $\mathbf{D}(t)$ (resp. $\mathbf{D}(x)$).*

For example, in our planning problem, an assignment $H = 2$ of the horizon h_v of a vehicle v induces the dynamic time variables $ts_{v,1}$, $ts_{v,2}$, $te_{v,1}$, and $te_{v,2}$ and the dynamic timeline variables $at_{v,1}$, $at_{v,2}$, $du_{v,1}$, and $du_{v,2}$. Concerning timeline at_v, variable $ts_{v,2}$ represents the temporal position of step 2 and variable $at_{v,2}$ the value of at_v at step 2.

It may be convenient to allow a time reference or a timeline to be initialized. This induces an additional dynamic variable indexed by 0. In our planning problem, we consider that timelines at_v are initialized.

Static constraints

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

Definition 6 *A static constraint c is simply a CSP constraint (Rossi, Beek, and Walsh 2006) whose scope is limited to static variables.*

In our planning problem, several static constraints can be defined. First, for each vehicle v, the horizon is equal to the number of visited areas plus one (Constraint 1) and the number of visited areas is equal to number of times variables vv_a take value v (Constraint 2 which uses the global CSP constraint *GCC*, for Global Cardinality Constraint). Second, the number of images taken by each vehicle is limited (Constraint 3 which uses the same *GCC* constraint). Last, the value of the criterion to be maximized is defined by Constraint 4, where the weighting factor $Md + 1$ ensures that the total number of visited areas has priority over the mission duration.

$$\forall v \in [1..Nv] : h_v = na_v + 1 \quad (1)$$
$$\forall v \in [1..Nv] : na_v = Card\{a \in A \,|\, vv_a = v\} \quad (2)$$
$$\forall v \in [1..Nv] : Ni_v \geq Card\{a \in Ai \,|\, vv_a = v\} \quad (3)$$
$$o = (\textstyle\sum_{v \in [1..Nv]} na_v) \cdot (Md + 1) - e \quad (4)$$

Dynamic constraints

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 to a definition of what is 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 $\langle SV, ST, SX, f\rangle$ where SV is a finite set of static variables, ST is a finite set of time references, SX is a finite set of 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 ST and SX. Variables in the scope of these induced CSP constraints must be either static variables in SV or dynamic variables in the set of dynamic variables induced by H for time references and timelines in ST and SX.*

In our planning problem, several dynamic constraints can be defined. For each vehicle v, Constraints 5 and 6 initialize timeline at_v and time reference ts_v. Constraints 7 to 9 link timeline du_v and time references ts_v and te_v. Constraints 10 and 11 enforce that landing occurs at the last step. Constraint 12 enforces that all the areas visited by v are different. Constraint 13 links timeline at_v and static variables vv. Constraint 14 produces a lower bound on the mission duration e which is greater than or equal to the current time plus the time v takes to go home. Constraint 15 prunes some suboptimal plans. Informally, it ensures that, if v visits areas a, a', a'', and a''' in the order $a \rightarrow a' \rightarrow a'' \rightarrow a'''$, then it must not be faster to visit them in the order $a \rightarrow a'' \rightarrow a' \rightarrow a'''$. Last, Constraint 16 defines the mission duration.

$\forall v \in [1..Nv]$

$$at_{v,0} = 0 \quad (5)$$
$$ts_{v,1} = 0 \quad (6)$$
$$\forall i \in [1..\mathbf{h}(v)] : du_{v,i} = Du(at_{v,i-1}, at_{v,i}) \quad (7)$$
$$\forall i \in [2..\mathbf{h}(v)] : ts_{v,i} = te_{v,i-1} \quad (8)$$
$$\forall i \in [1..\mathbf{h}(v)] : te_{v,i} = ts_{v,i} + du_{v,i} \quad (9)$$
$$\forall i \in [1..\mathbf{h}(v) - 1] : at_{v,i} \neq 0 \quad (10)$$
$$at_{v,\mathbf{h}(v)} = 0 \quad (11)$$
$$AllDifferent(at_{v,i} \,|\, i \in [1..\mathbf{h}(v)]) \quad (12)$$
$$\forall i \in [1..\mathbf{h}(v)], \forall a \in A : (at_{v,i} = a) \rightarrow (vv_a = v) \quad (13)$$
$$\forall i \in [1..\mathbf{h}(v)] : e \geq ts_{v,i} + Du(at_{v,i-1}, 0) \quad (14)$$
$$\forall i \in [3..\mathbf{h}(v)] : du_{v,i-2} + du_{v,i-1} + du_{v,i} \leq Du(at_{v,i-3}, at_{v,i-1}) + Du(at_{v,i-1}, at_{v,i-2}) + Du(at_{v,i-2}, at_{v,i}) \quad (15)$$
$$e = \max_{v \in [1..Nv]} te_{v,\mathbf{h}(v)} \quad (16)$$

Let us take some examples to illustrate Definition 7.

For each vehicle v, Constraint 7 is a dynamic constraint defined by the tuple $\langle \emptyset, \emptyset, \{at_v, du_v\}, f_{7,v}\rangle$ with $f_{7,v}$ the function which associates with each assignment H of $\mathbf{h}(v)$ the set of H ternary CSP constraints $\{du_{v,i} = Du(at_{v,i-1}, at_{v,i}) \,|\, i \in [1..H]\}$, each one connecting dynamic timeline variables $du_{v,i}$, $at_{v,i-1}$, and $at_{v,i}$. Similarly, Constraint 8 is defined by a tuple $\langle \emptyset, \{ts_v, te_v\}, \emptyset, f_{8,v}\rangle$ and Constraint 9 by a tuple $\langle \emptyset, \{ts_v, te_v\}, \{du_v\}, f_{9,v}\rangle$.

For each vehicle v, Constraint 12 is a dynamic constraint defined by the tuple $\langle \emptyset, \emptyset, \{at_v\}, f_{12,v}\rangle$ with $f_{12,v}$ the function which associates with each assignment H of $\mathbf{h}(v)$ the unique global CSP constraint *AllDifferent*$(\{at_{v,i} \,|\, i \in [1..H]\})$ which is a particular case of the *GCC* constraint.

For each vehicle v and each area a, Constraint 13 is a dynamic constraint defined by the tuple $\langle \{vv_a\}, \emptyset, \{at_v\}, f_{13,v,a}\rangle$ with $f_{13,v,a}$ the function which associates with each assignment H of $\mathbf{h}(v)$ the set of H binary CSP constraints $\{(at_{v,i} = a) \rightarrow (vv_a = v) \,|\, i \in [1..H]\}$, each one connecting dynamic timeline variable $at_{v,i}$ and static variable vv_a.

Constraint networks on timelines

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

Definition 8 *A constraint network N on timelines (CNT) is a tuple $\langle V, CS, T, X, CD\rangle$ 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 a time reference are totally and strictly ordered: $\forall t \in T, \forall i \in [1..\mathbf{h}(t)-1] : t_i < t_{i+1}$.

The CNT that results from the modeling of our planning problem is defined by the tuple $\langle V, CS, T, X, CD\rangle$, with $V = (\cup_{v=1}^{Nv}\{h_v, na_v\}) \cup (\cup_{a=1}^{Na}\{vv_a\}) \cup \{e, o\}$, $CS = \{c_i \,|\, i \in [1..4]\}$, $T = \cup_{v=1}^{Nv}\{ts_v, te_v\}$, $X = \cup_{v=1}^{Nv}\{at_v, du_v\}$, and $CD = \{c_i \,|\, i \in [5..16]\}$.

Assignments, solutions, consistency, and optimality

Definition 9 *An assignment of a CNT is an assignment of all its static variables (including all the horizon variables) and of all the induced dynamic variables.*

A solution of a CNT N 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.

A objective variable o is a static variable which is function of other static or dynamic variables and whose domain is equipped with a total order $\succ$. A solution S of N is optimal iff there is no other solution S' such that $S'_{\downarrow o} \succ S_{\downarrow o}$[2].

Figure 2 shows a example of solution of the CNT associated with the instance of Figure 1. The first table shows the activity plan of the first vehicle v: initially, v is at home; at time 0, it triggers the visit of area 1 which takes 7 units of time; then, it triggers successively the visit of areas 2 and 6 to finally go home. The second table shows the same kind of plan for the second vehicle w: horizons and times are different. The third table shows the assignment of static variables.

More generally, a planning problem can be cast as a CNT where timelines represent actions and states, and constraints model action feasibilities and state evolutions, as well as user objectives, and a valid plan can be extracted from any CNT solution.

step	0	1	2	3	4
start time ts_v	-	0	7	15	20
visited area at_v	0	1	2	6	0
duration du_v	-	7	8	5	8
end time te_v	-	7	15	20	28

step	0	1	2	3
start time ts_w	-	0	10	18
visited area at_w	0	4	7	0
duration du_w	-	10	8	9
end time te_w	-	10	18	27

$h_v = 4, h_w = 3, na_v = 3, na_w = 2$
$vv_1 = vv_2 = vv_6 = v, vv_4 = vv_7 = w, vv_3 = vv_5 = 0$
$e = 28, o = 127$

Figure 2: A solution of the CNT associated with the instance of Figure 1.

An Anytime Forward Search for CNTs

As said in the introduction, the objective of this paper is to define anytime algorithms for CNTs. To do this, the strategy we adopt is the same as the one used in (Pralet and Verfaillie 2008) to design optimal algorithms: to use standard CSP techniques and to adapt them to the specific features of planning problems. We first present the anytime CSP algorithm from which we start, and then its adaptation to CNTs.

A standard restarted depth-first search for CSPs

A standard restarted depth-first tree search (*restartedDFS*) for CSPs is given in Algorithm 1. This algorithm takes as an input a CSP defined by a set V of variables, a set C of constraints, and an objective $o \in V$ to be minimized. It performs a sequence of limited depth-first searches (DFS in short, line 6) until some stopping condition is met (line 4). Usually, this stopping condition is the fact that either a maximum CPU time has been reached or all the search tree has been traversed. Before each limited DFS, constraints are propagated in order to simplify the problem to be solved by removing inconsistent values from the domains of variables (line 5). After each limited DFS, a constraint is added to impose that each future solution is strictly better than the best solution found so far (line 7).

Each limited DFS is a depth-first search which can be restarted (line 20). A standard restart condition is the fact that a maximum number of backtracks has been reached. In function *limitedDFS*, if all the variables are assigned (line 12), the complete assignment is returned. Otherwise, an unassigned variable x is chosen (line 16), as well as a branching constraint c on x (line 17). Branching constraints are of the form $x\, cmp\, b$ with b a constant and cmp a comparator in $\{=, \neq, \leq, \geq, <, >\}$. If the restart condition is not met, constraint c is added to the current CSP, constraints are propagated (line 21) and, if no inconsistency is detected (line 22), the subtree associated with constraint c is explored (line 23). If a solution is returned, a new constraint is added to the problem to enforce that each future solution is strictly better than the best solution found so far (line 24). If the restart condition is not met, the subtree associated with the opposite constraint $\neg c$ is also explored (line 19).

Adaptation to CNTs

If only standard CSP settings are used in this algorithm, the results obtained on CNTs may be quite weak in terms of

[2] $A_{\downarrow v}$ denotes the value of variable v in assignment A.

Algorithm 1: A generic restarted DFS for solving CSPs

```
1  restartedDFS(V, C, o)
2  begin
3      A ← null
4      while ¬stopCondition() do
5          (V, C) ← propagate(V, C)
6          A' ← limitedDFS(V, C, o)
7          if A' ≠ null then (A, C) ← (A', C ∪ {o < A'_↓o})
8      return A
9  end
10 limitedDFS(V, C, o)
11 begin
12     if ∀x ∈ V, card(D(x)) = 1 then
13         return {(x, a) | x ∈ V, D(x) = {a}}
14     else
15         A ← null
16         x ← chooseUnassignedVar(V)
17         c ← chooseBranchingConstraint(x, V, C)
18         (c1, c2) ← (c, ¬c)
19         for i = 1 to 2 do
20             if ¬restartCondition() then
21                 (V', C') ← propagate(V, C ∪ {ci})
22                 if ∀x ∈ V', D(x) ≠ ∅ then
23                     A' ← limitedDFS(V', C', o)
24                     if A' ≠ null then (A, C) ← (A', C ∪ {o < A'_↓o})
25         return A
26 end
```

computation time and plan quality. A usual way of overcoming such a difficulty is to design specific parameter settings for each particular planning problem to be solved. In this paper, another way is proposed: the definition of generic parameter settings allowing features of CNTs to be exploited. Parameters to be set are functions *propagate*, *chooseUnassignedVar*, *chooseBranchingConstraint*, and *restartCondition*. All the mechanisms described thereafter have been implemented in the so-called *restartedDFS(CNT)* algorithm which is the core of the SCOT CNT solver.

First CSP encoding of CNTs and constraint propagation At any step, the *restartedDFS(CNT)* algorithm reasons on a current CSP (V, C). Following the *lazy* approach used in (Pralet and Verfaillie 2008), given a CNT $N = (V', CS, T, X, CD)$, at any step, the set V of CSP variables to be considered is $V' \cup \{x_i \mid (x \in T \cup X) \wedge (i \in [1..\min(\mathbf{D}(h(x)))])\}$ *i.e.*, the set of variables that are necessarily present in any complete assignment of N due to the minimum values in the domains of the horizon variables. The set C of CSP constraints to be considered is obtained *via* a function *extend* which produces a set of constraints that must be necessarily satisfied by any complete assignment of N. For instance, for dynamic constraint 8 in our illustrative example ($\forall i \in [2..h_v] : ts_{v,i} = te_{v,i-1}$), when the minimum value of h_v is 3, function *extend* returns two constraints ($ts_{v,2} = te_{v,1}$ and $ts_{v,3} = te_{v,2}$).

Function *propagate* needs to be adapted so that function *extend* is called whenever the minimum value in the domain of a horizon variable is increased. As a result, function *propagate* adapted to CNTs can add new variables and constraints to the current CSP.

Variable choice Concerning the choice of the next variable to assign (function *chooseUnassignedVar*), a limitation of standard CSP algorithms is that they do not reason in terms of time and sequences of chronologically ordered variables, but only in terms of variables and constraints, even though many efficient anytime planning algorithms use a forward search approach in which decisions are made chronologically (Hoffmann and Nebel 2001).

The CNT framework offers the possibility to use both constraints and forward search. Although intuitive, a forward search in the CNT framework requires some preliminary definitions to be formally defined, mainly because of the presence of concurrent time references. The basic idea is to associate with each time reference t a so-called current index, denoted $\mathbf{ci}(t)$, which is the smallest index $i \leq \min(\mathbf{D}(\mathbf{h}(t)))$ at which either t_i is not assigned, or there exists a timeline x whose time reference is t such that x_i is not assigned ($\mathbf{ci}(t) = +\infty$ when such an index does not exist). Then, the so-called current time $\mathbf{ct}(t)$ of a time reference t can be defined as the minimum value in the domain of dynamic time variable $t_{\mathbf{ci}(t)}$ ($\mathbf{ct}(t) = +\infty$ when $\mathbf{ci}(t) = +\infty$). From this, the current time $\mathbf{ct}(N)$ of a CNT N can be defined as the minimum time over the current times of all the time references in N. Informally speaking, the current time of a CNT is the first time at which a decision must be made. In the forward approach, when current time $\mathbf{ct}(N)$ is different from $+\infty$, the only variables that can be chosen by function *chooseUnassignedVar* are:

- non assigned dynamic timeline variables $x_{\mathbf{ci}(\mathbf{t}(x))}$ such that $\mathbf{ct}(N)$ is the only possible value in the domain of the dynamic time variable $\mathbf{t}(x)_{\mathbf{ci}(\mathbf{t}(x))}$;
- if no such variable exists, non assigned dynamic time variables $t_{\mathbf{ci}(t)}$ such that $\mathbf{ct}(t) = \mathbf{ct}(N)$.

If $\mathbf{ct}(N) = +\infty$, any non assigned static variable can be chosen. When several variables can be selected by the forward approach, classical CSP variable choice heuristics can be used to select one of them. Informally speaking, the forward approach chooses to assign first dynamic timeline variables that are necessarily associated with the current time of the CNT, then dynamic time variables that may be associated with the current time, and finally static variables.

Branching constraint choice As action choice heuristics are crucial in classical planning, parameter settings for function *chooseBranchingConstraint* are crucial for CNT solving. Several options have been implemented in SCOT:

- random approach: a branching constraint $x = b$ is chosen in a completely random way over all values $b \in \mathbf{D}(x)$;
- constraint propagation approach: for each value $b \in \mathbf{D}(x)$, constraint $x = b$ is added, constraint propagation is performed, and the lower bound $\mathbf{bo}(b)$ on the objective is recorded; then, a value b that minimizes $\mathbf{bo}(b)$ is

selected, ties being broken randomly;

- simulation approach: several stochastic greedy searches are triggered, starting from the current CNT; then, the first value in one of the best trajectories is selected;
- hand-defined heuristics: the CNT framework allows *efficient* models to be defined, involving constraints that are specific to the problem to be solved and help to prune the search tree; the same way, it is possible to define search heuristics that are specific to the problem to be solved and help to guide the search towards high quality plans;
- hand-defined stochastic heuristics: because heuristics may fail, it may be useful to include a stochastic aspect in hand-defined heuristics, leading to branching schemes such as "choose $x = b$ with probability 0.8 and $x \neq b$ with probability 0.2".

Restart condition In a standard restarted depth-first search, restart conditions described in function *restartCondition* often specify that a restart must be performed when a maximum number of backtracks is reached. However, in CNTs, we often observe several constraints connecting variables associated with the same temporal position. These constraints connect for example decision variables that together define a global action. As a result, it may be worth spending some time searching for a consistent assignment of these variables. To do that, an option consists in specifying restart conditions not in terms of a maximum number k of backtracks, but in terms of a maximum number k of temporal positions on which it is possible to go back. For example, if $k = 0$, backtracking to a previous temporal position is forbidden. As done in several CSP solvers such as CP Optimizer[3], parameter k can be geometrically increased search after search.

It is worth noting that, through the use of various restart schemes, the *restartedDFS(CNT)* algorithm covers both complete tree search, when no restart is performed, and iterated greedy stochastic search, when no backtrack is allowed, as well as many intermediate search schemes.

Back to the CSP encoding of CNTs and to constraint propagation One of the key points in the anytime forward approach we defined is how CNTs are encoded as CSPs. In most of the CSP-based approaches to planning (van Beek and Chen 1999; Do and Kambhampati 2000), the model is unfolded over a given number of steps. In (Pralet and Verfaillie 2008), as explained above, it is unfolded too over a number of steps which is variable and may increase during the search. In both cases, this unfolding may generate very large CSPs. On some problems, we observed that just creating the unfolded problem may take several tens of seconds or several minutes. For some problems, there is even not enough memory for the unfolded problem to be created without memory swaps. Such a situation is incompatible with anytime requirements.

To answer this point, we propose a new CSP encoding of CNTs which is able to reason only on a problem slice, that is to reason by considering simultaneously only a small number of consecutive steps. In its current form, this encoding is defined for problems in which dynamic constraints link timelines and time references sharing a same horizon variable h and have the form $\forall i \in [a..h - b] : c_i$, with c_i a stationary dynamic constraint, that is a dynamic constraint whose definition does not depend on the particular step i considered. Dynamic constraint 7 in our illustrative example ($\forall i \in [1..\mathbf{h}(v)] : du_{v,i} = Du(at_{v,i-1}, at_{v,i})$) is an example of stationary dynamic constraint. Roughly speaking, the idea of the automatic encoding is to create variables denoted $du_v^{(0)}$, $at_v^{(-1)}$, and $at_v^{(0)}$ which represent respectively the value of timeline du_v at the current step and the value of timeline at_v one step before the current step and at the current step, and to impose constraint $du_v^{(0)} = Du(at_v^{(-1)}, at_v^{(0)})$. Therefore, dynamic constraint 7 is represented at any step of the algorithm by three CSP variables and one CSP constraint instead of being represented by $2 \cdot \min(\mathbf{D}(\mathbf{h}(v))) + 1$ CSP variables and $\min(\mathbf{D}(\mathbf{h}(v)))$ constraints. Such an encoding is correct thanks to the forward approach. Note that the planning problem for a team of UAVs, used as an illustrative example, can be modeled using such a slice approach and that this approach can be extended to other kinds of constraints.

The price to pay for this compact encoding is that constraint propagation algorithms must be adapted in order to be able to reason only on a problem slice. This adaptation has been achieved in SCOT on top of *Choco 2.0*. See (Pralet and Verfaillie 2009) for more details.

Experiments

Experiments were performed on three domains: BlocksWorld (IPC2), Satellite (IPC3, *propositional*, *simpletime*, and *time* versions), and Trucks (IPC5, *propositional* and *time* versions). For space limitations reasons, only results on BlocksWorld, Satellite *propositional*, Satellite *time*, and Trucks *time* are presented. CNT models have been built for these domains, using a language which extends to CNTs the usual way of defining constraints in *Choco 2.0*. For instance, a constraint like $\forall i \in [2..h_v] : ts_{v,i} = te_{v,i-1}$ is expressed *via* the following Java code: `postForall(2,h[v],eq(ts[v],offset(te[v],-1)));`. Developing a higher level language to express CNT models in a more natural way is out of the scope of this paper. Experiments were run on a SunUltra45, 1.6GHz, 1GBRAM.

Analysis of various parameter settings

The goal of the first experiments performed is to assess the influence of various parameter settings of the *restartedDFS(CNT)* algorithm on the quality profile. In Figure 3, we present some results obtained on two particular instances: instance *satellite-time-8* in the first row and instance *trucks-time-8* in the second one. In both cases, the quality is the makespan to be minimized. Each column corresponds to the study of a particular parameter: CSP encoding and variable choice in the first column, restart condition in the second one, and branching constraint choice in the third one. Because stochastic choices are made, each parameter conf-

[3]CP Optimizer: http://www.ilog.com/products/cpoptimizer/.

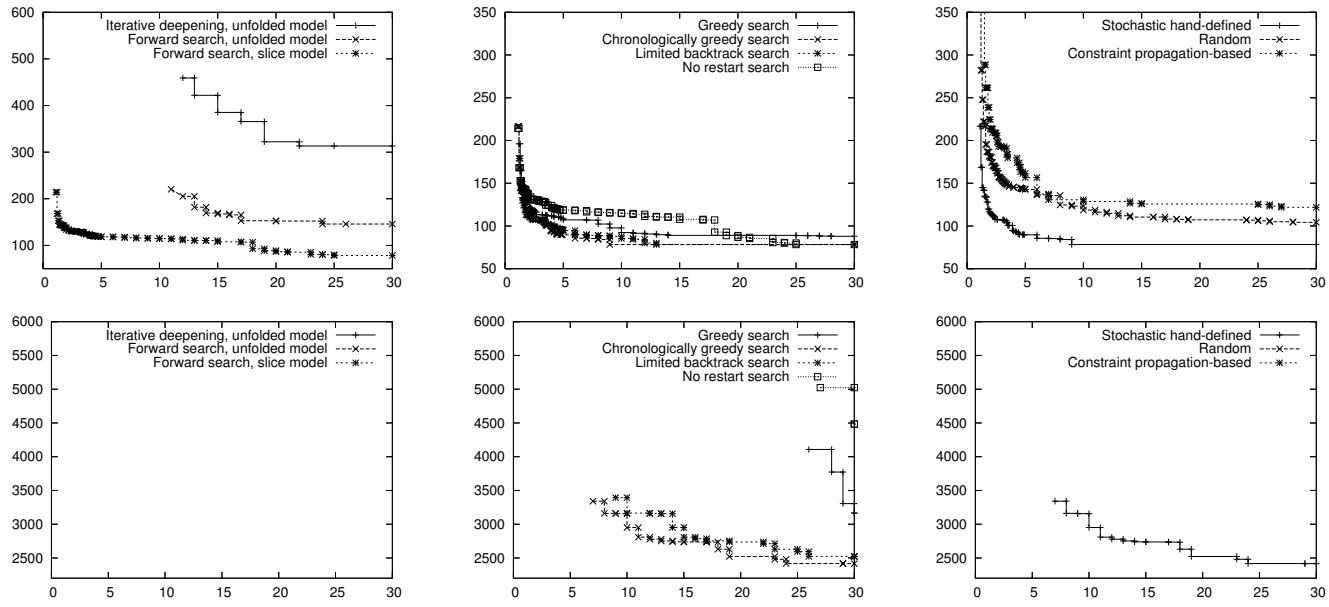

Figure 3: Median quality evolution on instances *satellite-time-8* (first row) and *trucks-time-8* (second row). Variable choice and CSP encoding (first column). Restart condition (second column). Branching constraint choice (third column). Time in seconds.

iguration is run 50 times on each instance with a time limit of 30 seconds per run. The plots correspond to a median quality evolution: a point (t, q) means that, at time t, half of the runs got a quality greater than or equal to q.

Variable choice and CSP encoding In the first column, we compare three approaches: (1) iterative deepening (where dimension variables are assigned first and then other variables in any order, not necessarily chronological), (2) forward search (as described in paragraph "Variable choice" in the previous section) on top of an unfolded model, and (3) forward search on top of a slice model. Fixed parameter settings are no restart and random branching constraint choice. Plots in the first row clearly show the benefit of a forward search with regard to an iterative deepening one and the benefit of a slice model with regard to an unfolded one. However, the second row shows that, on the second instance, none of the tested configurations manages to find a plan within the time limit. This leads us to look at other parameters.

Restart condition In the second column, we compare four restart approaches: (1) completely greedy search, restarting each time a backtrack is needed, (2) chronologically greedy search, restarting each time a backtrack to a previous temporal position is needed, (3) limited backtrack search, restarting when a maximum number k of backtracks is reached, with k geometrically increased restart after restart, and (4) no restart search. Fixed parameter settings are forward search, slice encoding, and stochastic hand-defined heuristics. Both rows (the second one more clearly than the first one) show the superiority of the second and third approaches with regard to the first and fourth ones which are too diversified for the former and not enough diversified for the latter.

Branching constraint choice Last, in the third column, we compare three branching heuristics: (1) stochastic hand-defined heuristics, (2) completely random choice, and (3) constraint propagation-based heuristics. Fixed parameter settings are forward search, slice encoding, and limited backtrack search. Not surprisingly, both rows show the superiority of stochastic hand-defined heuristics. On the second instance, they are the only ones able to produce plans within the time limit. Constraint propagation used as heuristics seems to be time consuming and not informative enough.

Comparison with existing planners

Other experiments were performed to compare SCOT with other planners. For that, we selected:

- one domain-independent planner: SGPlan6 (Hsu and Wah 2008), winner of the *temporal satisficing* track of the last planning competition (IPC6);
- two domain-dependent planners: TALplanner, winner of the domain-dependent track in IPC2, and TLplan, winner of the same track in IPC3; therefore, we used TALplanner for domain BlocksWorld (IPC2) and TLplan for domain Satellite (IPC3); no TALplanner or TLplan model is available for domain Trucks; moreover, because TALplanner code is not available on the web, results were collected from the IPC2 site.

It must be stressed that none of these planners has been built to be anytime, but to produce as quickly as possible a plan of as high as possible quality.

For each algorithm, the time limit is of 120 seconds. The parameter settings used for the *restartedDFS(CNT)* algorithm in SCOT are forward search, slice encoding, lim-

Instance	SGPlan6	TALplanner TLplan	SCOT-first	SCOT-best	SCOT-opt
B-10-0	38 (0.3)	34 (0.3)	34 (1.1)	34 (1.1)	(1.3)
B-11-0	106 (5.3)	32 (0.2)	32 (1.1)	32 (1.1)	(1.1)
B-12-0	48 (10.4)	40 (0.2)	34 (1.2)	34 (1.2)	(1.2)
B-13-0	58 (0.5)	44 (0.2)	44 (1.3)	42 (1.8)	(2.8)
B-14-0	- (-)	40 (0.2)	38 (1.3)	38 (1.3)	(1.3)
B-15-0	102 (0.8)	48 (0.2)	40 (1.3)	40 (1.3)	(1.3)
B-17-0	- (-)	50 (0.2)	46 (1.4)	46 (1.4)	(1.4)
B-18-0	164 (143.9)	60 (0.2)	58 (1.7)	58 (1.7)	(1.7)
B-19-0	- (-)	62 (0.2)	62 (1.8)	62 (1.8)	(1.8)
B-20-0	106 (42.8)	64 (0.2)	60 (1.8)	60 (1.8)	(1.8)
B-25-0	- (-)	84 (0.2)	82 (2.4)	82 (2.4)	(2.4)
B-28-0	- (-)	96 (0.2)	92 (2.8)	92 (2.8)	(2.8)
S-P-05	16 (0.6)	10 (0.02)	9 (0.9)	7 (0.9)	(4.8)
S-P-06	24 (0.6)	11 (0.02)	11 (0.9)	8 (0.9)	(1.2)
S-P-07	22 (0.8)	10 (0.02)	9 (1.0)	6 (1.1)	(1.6)
S-P-08	26 (0.9)	11 (0.02)	11 (1.0)	8 (1.1)	(21.4)
S-P-09	34 (0.1)	10 (0.02)	9 (1.2)	6 (41.1)	(41.2)
S-P-10	35 (0.1)	10 (0.02)	10 (1.3)	8 (1.4)	?
S-P-11	34 (0.1)	12 (0.03)	12 (1.2)	8 (2.7)	(4.9)
S-P-12	72 (0.3)	17 (0.04)	16 (1.9)	14 (2.7)	?
S-P-13	60 (0.4)	18 (0.05)	18 (2.0)	13 (5.9)	?
S-P-14	44 (0.3)	12 (0.03)	11 (1.7)	8 (10.3)	(65.8)
S-P-15	51 (0.4)	14 (0.04)	13 (2.4)	8 (108.4)	?
S-P-16	54 (0.6)	11 (0.04)	11 (3.4)	7 (3.6)	?
S-T-05	394.3 (0.6)	170.3 (0.02)	162.4 (1.0)	105.3 (13.5)	(13.7)
S-T-06	550.1 (0.6)	140.1 (0.02)	124.3 (1.0)	68.8 (1.9)	(1.9)
S-T-07	349.7 (0.9)	121.2 (0.03)	111.2 (1.9)	60.3 (4.9)	(26.9)
S-T-08	413.2 (0.1)	79.1 (0.04)	77.5 (2.2)	74.3 (20.4)	?
S-T-09	888.7 (0.2)	127.6 (0.05)	123.7 (1.5)	86.2 (4.6)	(31.5)
S-T-10	921.1 (0.2)	209.6 (0.07)	192.5 (1.4)	117.4 (87.9)	(103.8)
S-T-11	873.6 (0.2)	174.10 (0.06)	165.9 (1.5)	128.1 (78.9)	?
S-T-12	1452.4 (0.4)	236.5 (0.1)	233.2 (1.9)	139.10 (85.3)	?
S-T-13	971.7 (0.7)	210.4 (0.1)	191.4 (2.2)	127.7 (95.3)	(112.7)
S-T-14	727.6 (0.5)	175.8 (0.1)	159.1 (1.9)	105.7 (12.8)	(114.2)
S-T-15	852.9 (0.8)	169.3 (0.2)	165.10 (2.5)	103.2 (94.7)	?
S-T-16	1216.5 (1.0)	178.9 (0.3)	150.3 (3.5)	92.5 (77.6)	?
T-T-01	843.3 (0.3)	not run	843.3 (1.7)	843.3 (1.7)	(22.5)
T-T-02	1711.5 (0.5)	not run	1711.5 (2.0)	1711.5 (2.0)	?
T-T-03	1470.2 (0.6)	not run	1470.1 (2.7)	1202.6 (4.2)	?
T-T-04	2629.5 (0.8)	not run	2629.5 (4.2)	2629.5 (4.2)	?
T-T-05	2250.9 (0.8)	not run	2250.9 (3.5)	1671.6 (4.2)	?
T-T-06	5021.1 (0.1)	not run	4774.2 (4.0)	4319.0 (5.5)	?
T-T-07	1878.2 (0.9)	not run	1854.3 (13.2)	1633.3 (60.7)	?
T-T-08	2303.1 (0.2)	not run	- (-)	2417.0 (55.2)	?
T-T-09	4835.7 (0.2)	not run	3089.2 (10.5)	2667.2 (99.5)	?
T-T-10	3034.5 (0.2)	not run	3020.1 (29.4)	2759.1 (97.2)	?
T-T-11	3492.1 (0.2)	not run	3225.10 (9.2)	2305.5 (108.6)	?
T-T-12	4543.1 (0.3)	not run	3990.9 (10.3)	3024.7 (74.9)	?

Table 1: Comparison between SCOT and existing planners. "B" stands for *BlocksWorld*, "S-P" for *Satellite-Propositional*, "S-T" for *Satellite-Time*, and "T-T" for *Trucks-Time*.

ited backtrack search, and hand-defined stochastic heuristics. Because of its stochastic features, SCOT is run 20 times and the median quality evolution is recorded. In Table 1, for each instance, we display in the first two columns elements $q(t)$ where q is the quality produced by SGPlan6, TALplanner, or TLplan and t is the associated *cpu* time. Then, column *SCOT-first* displays elements $q(t)$ which indicate that, at time t, SCOT produces a plan of quality q which is better than or equal to the quality produced by all the other planners. The same way, column *SCOT-best* displays elements $q(t)$ which indicate that the best plan produced by SCOT is of quality q and that it is produced at time t. Last, column *SCOT-opt* displays elements (t) or ? where (t) indicates that optimality is proved at time t and ? that it is not established within the time limit.

These results show that, on almost all the instances, SCOT manages in a few seconds to produce plans whose quality is better than or equal to the quality of the plans produced by all the other planners. Mainly in the last three domains, it is moreover able to improve on the quality of the first plan found by a significant factor. Mainly in the first three domains, it is even able to prove the optimality of the best plan found. On the negative side, it must be emphasized that the time needed to produce the first plan is always several times larger than with SGPlan6, TALplanner, or TLplan. One of the reasons for this is, even with an efficient CSP encoding, the time needed to create the CSP and all its data structures which may represent half of the total *cpu* time.

Conclusion

This paper shows that it is possible to define an efficient generic anytime forward-search algorithm on top of the CNT framework. It shows also how the CNT framework and the associated SCOT solver together allow user knowledge of a specific planning problem to be incorporated in the form of constraints, heuristics, or algorithm parameters, in order to solve it still more efficiently. For future work, an immediate objective is to reduce the time needed to produce a first plan.

References

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence* 16:123–191.

Do, M., and Kambhampati, S. 2000. Solving Planning-Graph by Compiling it into CSP. In *Proc. of AIPS-00*, 82–91.

Frank, J., and Jónsson, A. 2003. Constraint-Based Attribute and Interval Planning. *Constraints* 8(4):339–364.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

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

Hsu, C., and Wah, B. 2008. The SGPlan6 Planning System in IPC6. In *Proc. of the International Planning Competition (IPC6)"*.

Kvarnström, J., and Doherty, P. 2001. TALplanner: A Temporal Logic Based Forward Chaining Planner. *Annals of Mathematics and Artificial Intelligence* 30:119–169.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Pralet, C., and Verfaillie, G. 2008. Using Constraint Networks on Timelines to Model and Solve Planning and Scheduling Problems. In *Proc. of ICAPS-08*, 272–279.

Pralet, C., and Verfaillie, G. 2009. Slice Encoding for Constraint-based Planning. In *Proc. of CP-09*.

Rossi, R.; Beek, P. V.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*. Elsevier.

van Beek, P., and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. In *Proc. of AAAI-99*, 585–590.

Verfaillie, G.; Pralet, C.; and Lemaître, M. 2008. Constraint-based Modeling of Discrete Event Dynamic Systems. *Journal of Intelligent Manufacturing, Special Issue on "Planning, Scheduling, and Constraint Satisfaction"*. Published online.

Zilberstein, S. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17(3):73–83.