Using constraint networks on timelines to model and solve planning and scheduling problems

Paper #34

Abstract

In the last decades, there has been an increasing interest in the connection between planning and constraint programming. Several approaches were used, leading to different forms of combination between the two domains. In this paper, we present a new framework, called Constraint Network on Timelines (CNT), to model and solve planning and scheduling problems. Basically, CNTs are a kind of dynamic CSPs, enhanced with special variables called dimension variables representing the initially unknown number of steps in a valid or optimal plan. We also present an algorithm and experimental results showing that the expressiveness of CNTs allows efficient models to be developed, and can lead to significant gains on problems taken from planning competitions.

Introduction

In the last decades, there has been an increasing interest in the connection between planning and Constraint Programming (CP). As already recognized in (Nareyek *et al.* 2005), this interest has led to three main kinds of combination between planning and CP.

1. CP can be used as a plug-in to solve efficiently subproblems generated during planning, e.g. to check whether resource constraints may be violated in the future. This plug-in approach allows existing planners to be enhanced with CP techniques but does not exploit all the capabilities of constraint programming.
2. In approaches inspired from (Kautz & Selman 1992), a CSP (Constraint Satisfaction Problem (Dechter 2003)) or a dynamic CSP (Mittal & Falkenhainer 1990) is built to solve the planning problem over a fixed horizon k, which is incremented if no plan is found. This technique is used in CPlan (van Beek & Chen 1999), GP-CSP (Do & Kambhampati 2001), and CSP-Plan (Lopez & Bacchus 2003). The size-bounded CSPs to be solved are built either from a planning graph (Blum & Furst 1997) or directly from STRIPS or PDDL representations (Fikes & Nilsson 1971; McDermott 1998). They contain variables representing the state and the actions at each step $i \in [1..k]$, and constraints specifying the initial and goal states, action preconditions, and action effects. Other constraints may be added manually. This approach can be very efficient but the CSPs built can become too large because all variables are duplicated at each step.
3. Other approaches tackle planning problems as a kind of dynamic CSP without fixing the horizon. This includes planners like *CPT* (Vidal & Geffner 2006), which represents the planning problem by a set of temporal variables associated with actions and action preconditions, and by a set of temporal constraints. The associated CSP is directly obtained from PDDL descriptions. It is dynamic in the sense that at each step of the search, variables and constraints are active or not. (Nareyek 2001) proposes another approach, which involves a CSP with a dynamic and constrained graphical structure. Another example is Constraint-based Attribute and Interval Planning (Frank & Jónson 2003), whose principle is to add to a current incomplete plan so-called intervals. The latter represent that some predicate holds over a time slot $[t_s, t_e]$, and must satisfy some compatibility constraints.

In this paper, we propose a new generic constraint-based approach to model and solve planning and scheduling problems. This approach, called *Constraint Network on Timelines* (CNT), is included in the third category but covers all approaches in the second one. A CNT is a kind of dynamic CSP, in which the dynamic aspect comes from the explicit presence of *dimension variables* representing the possibly unknown number of steps in the plans sought. The presentation here differs from the first version of CNTs introduced in (Anonymous 2008a). The paper is organized as follows. We first present the CNT framework and compare it with existing approaches. The different modeling capabilities of CNTs are illustrated on some planning problem examples taken from international planning competitions (IPCs). We then present an algorithm to seek for plans and optimal plans from CNT representations, highlighting the special role of dimension variables. Last, we give experimental results on IPC problems. These results show that using the modeling capabilities of CNTs can lead to significant gains in computation time. In particular, some problems unsolved by existing optimal planners are solved optimally in a few seconds, thanks to the expressiveness of CNTs, which allows a variety of information to be modeled.

Constraint networks on timelines

In the following, we denote by $[a..b]$ the set of integers between a and b, and given a variable x, we denote by $d(x)$ its domain of values. In order to illustrate the framework, we consider a space application example inspired from (Anonymous 2008b).

A satellite must download a set of N_O observations down to Earth over a time period $[STA, END]$. Over this time period, it goes through a set $\{[SE_k, EE_k], k \in [1..N_E]\}$ of N_E eclipse periods, and a set $\{[SD_k, ED_k], k \in [1..N_D]\}$ of N_D station visibility windows, during each of which a block of observations can be downloaded. Downloading observation $o \in [1..N_O]$ takes a duration D_o. At any time, the level of energy available on board is somewhere between $ENmin$ and $ENmax$. Its evolution depends on the power P_{dl} consumed when a download is performed, the power P_{sat} consumed by the platform, and the power P_{sol} produced by solar panels when the satellite is not in eclipse. The goal is to download all observations while respecting limitations on the level of energy. The initial level of energy, at time STA, is denoted $ENinit$.

To model this problem, we can first define a set of "classical" variables: for each observation o, we introduce one variable nd_o of domain $d(nd_o) = [0..N_D]$ to represent the index k of the download slot during which observation o is downloaded (value 0 if o is not downloaded), and for each download slot index $k \in [1..N_D]$, we introduce variables sd_k and ed_k, of domain $d(sd_k) = d(ed_k) = [SD_k, ED_k]$ to represent respectively the start and end times of the download occurring during download slot k. To model the evolution of the energy level without fixing the number of steps in this evolution, we need to introduce a set of variables en_i whose cardinality is unknown. This will be possible in the CNT framework thanks to the notion of timeline.

Definition 1 *(Timeline) A timeline tl is a pair $tl = (d(tl), h(tl))$ where $d(tl)$ is a set of values and $h(tl)$ is a variable whose domain of values $d(h(tl))$ is included in $\mathbb{N}$. $d(tl)$ is called the domain of tl and $h(tl)$ its dimension (dimension is denoted h like horizon).*

Definition 2 *(Variables associated with a timeline) Given an assignment A of $h(tl)$, a timeline $tl = (d(tl), h(tl))$ defines a finite set of variables $V(tl, A) = \{tl_i \mid i \in [1..A]\}$, whose domain of values is $d(tl_i) = d(tl)$. This set is empty if $h(tl)$ takes value 0. In order to distinguish variables defined by a timeline from classical variables, variables in $V(tl, A)$ are called timeline-variables, or more shortly* t-variables*.*

The maximal set of t-variables which may be defined by a timeline is $\{tl_i \mid i \in [1..max(d(h(tl)))]\}$. This set can be infinite if $d(h(tl))$ is not bounded. Among t-variables in this set, the ones in $\{tl_i \mid i \in [1..min(d(h(tl)))]\}$ are *mandatory*, because they exist whatever the timeline dimension is. The others are *optional*.

Definition 3 *(Assignment of a set of timelines) Let T be a set of timelines. An assignment A of T is the union of an assignment A_H of all the dimension variables of the timelines in T, and of an assignment A_V of all t-variables in $\cup_{tl \in T} V(tl, A_H[h(tl)])$, where $A_H[h(tl)]$ denotes the assignment of $h(tl)$ in A_H.*

Let us illustrate these notions on the satellite example. We can first introduce one dimension variable h of domain $d(h) = [1..\infty]$ to represent the number of important steps in the evolution of the level of energy. We use four timelines, $t = ([STA, END], h)$, $en = ([ENmin, ENmax], h)$, $ec = (\{0,1\}, h)$, and $dl = (\{0,1\}, h)$, to represent respectively the time associated with each step, the current level of energy, the current eclipse status, and the current download status. Given an assignment A of h, these timelines induce a set of t-variables $\{tl_i \mid tl \in \{t, en, ec, dl\}, i \in [1..A]\}$, where tl_i represents the value of timeline tl at step i. A tabular representation of an assignment of the different timelines is given below. The first column means that at $t_1 = 0$, the energy level is $en_1 = 300$, the satellite is in eclipse ($ec_1 = 1$), and it is not downloading observations ($dl_1 = 0$). At step 2, at $t_2 = 30$, the energy level has decreased to $en_2 = 270$, and the satellite starts downloading observations ($dl_2 = 1$). This download ends at step 3 ($dl_3 = 0$), at $t_3 = 48$. At step 4, at $t_4 = 150$, the satellite is not is eclipse anymore ($ec_4 = 0$). And so on until step $h = 8$.

	1	2	3	4	5	6	7	8
t	0	30	48	150	235	248	272	350
en	300	270	216	114	284	271	223	145
ec	1	1	1	0	0	1	1	1
dl	0	1	0	0	1	1	0	0

Variables defined by timelines must usually satisfy some constraints. We therefore introduce the notion of constraint on timelines.

Definition 4 *(Constraint) A classical CSP constraint c is defined by a pair $(S(c), R(c))$ where $S(c)$ (the scope of c) is the finite set of variables over which the constraint holds, and $R(c)$ (the relation associated with c) is any explicit or implicit representation of the set of allowed combinations of values of the variables in $S(c)$.*

Definition 5 *(Constraint on timelines) A constraint on timelines is a triple $c = (S_V(c), S_T(c), fct(c))$ where $S_V(c)$ is a finite set of variables, $S_T(c)$ is a finite set of timelines, and $fct(c)$ is a function which associates a finite set of CSP constraints with each assignment A of the dimension variables of the timelines in $S_T(c)$. It is moreover assumed that the scope of each of the CSP constraints in $fct(c)(A)$ is included in $S_V(c) \cup \{tl_i \mid tl \in S_T(c), i \in [1..A[h(tl)]]\}$.*

Given a timeline tl and a variable x, an example of constraint on timelines is $c : \forall i \in [1..h(tl) - 1], tl_{i+1} \neq tl_i + x$. Implicitly, c is the triple $(S_V(c), S_T(c), fct(c))$ where $S_V(c) = \{x\}$, $S_T(c) = \{tl\}$, and $fct(c)$ is the function which associates, with each assignment A of $h(tl)$, the set of CSP constraints $\{tl_{i+1} \neq tl_i + x \mid i \in [1..A-1]\}$. Another example of constraint on timelines is: $\exists i \in [2..h(tl)], tl_i = 0$. It corresponds to the triple $(\varnothing, \{tl\}, fct)$, where fct associates, with each assignment A of $h(tl)$, the CSP constraint $\exists i \in [2..A], tl_i = 0$. In order to impose conditions on the final state of a timeline, constraints of the form $tl_{h(tl)} = g$ can be considered. They associate, with each assignment A of $h(tl)$, the CSP constraint $tl_A = g$. We can also define constraints like $alldifferent(\{tl_i \mid i \in [1..h]\})$, which implicitly associates with each assignment A of h the classical

CSP constraint $\mathit{alldifferent}(\{tl_i \mid i \in [1..A]\})$. Such constraints, which are usually called global constraints because they hold on a large number of variables, are interesting because they can be handled by dedicated powerful inference algorithms available in constraint programming tools.

To be more concrete, the satellite problem can be modeled by constraints c_1 to c_8 listed below. Constraint c_1 defines the initial state. Constraint c_2 expresses that the end time of a download is its start time plus its duration. Constraints c_3 to c_6 define the evolution of the system state (ec, dl, en, t) at any step i: c_3 and c_4 define when the satellite is in an eclipse state and in a download state, c_5 defines the evolution of the level of energy, and c_6 asserts that the time t_{i+1} of step $i+1$ is the minimum of all important time points strictly greater than t_i. Last, c_7 defines a condition on the final state and c_8 corresponds to the problem goal. Note that the domain of values associated with timelines also enforce constraints. For instance, the domain of value $d(en) = [ENmin, ENmax]$ of timeline en imposes a constraint on the minimum level of energy.

$$(t_1 = STA) \wedge (en_1 = ENinit) \qquad (c_1)$$

$$\forall k \in [1..N_D], ed_k = sd_k + \textstyle\sum_{o \in [1..N_O] \mid nd_o = k} D_o \qquad (c_2)$$

$$\forall i \in [1..h],$$

$$(ec_i = 1) \leftrightarrow (\exists k \in [1..N_E], SE_k \leq t_i < EE_k) \qquad (c_3)$$

$$(dl_i = 1) \leftrightarrow (\exists k \in [1..N_D], sd_k \leq t_i < ed_k) \qquad (c_4)$$

$$\forall i \in [1..h-1],$$

$$en_{i+1} = \min(ENmax, en_i + (t_{i+1} - t_i) \cdot P) \qquad (c_5)$$
$$\text{with } P = (1 - ec_i) \cdot P_{sol} - dl_i \cdot P_{dl} - P_{sat}$$

$$t_{i+1} = \min\{t \in T_D \cup T_E \cup \{END\} \mid t > t_i\} \qquad (c_6)$$
$$\text{with } T_D = \cup_{k \in [1..N_D], sd_k < ed_k} \{sd_k, ed_k\}$$
$$\text{and } T_E = \cup_{k \in [1..N_E]} \{SE_k, EE_k\}$$

$$t_h = END \qquad (c_7)$$

$$\forall o \in [1..N_O], nd_o \neq 0 \qquad (c_8)$$

All notions defined previously are assembled in the notion of *constraint network on timelines* (CNT).

Definition 6 *(Constraint network on timelines) A constraint network on timelines cnt is defined by a tuple $cnt = (V, C_V, T, C_T)$, where V is a finite set of variables, C_V is a finite set of constraints whose scopes are included in V, T is a finite set of timelines whose dimensions are included in V, and C_T is a finite set of constraints on timelines (S_V, S_T, fct) such that $S_V \subset V$ and $S_T \subset T$.*

The satellite download problem can be modeled by the CNT (V, C_V, T, C_T), where $V = \{nd_o \mid o \in [1..N_O]\} \cup \{sd_k \mid k \in [1..N_D]\} \cup \{ed_k \mid k \in [1..N_D]\} \cup \{h\}$, $C_V = \{c_2, c_8\}$, $T = \{t, en, ec, dl\}$, and $C_T = \{c_1, c_3, c_4, c_5, c_6, c_7\}$.

Among the various problems which can be formulated on CNTs, a useful one is to seek for a consistent assignment:

Definition 7 *(Consistent assignment of a CNT) A consistent assignment (a solution) of a constraint network on timelines $cnt = (V, C_V, T, C_T)$ is an assignment of the variables in V and of the timelines in T such that all CSP constraints in C_V and all CSP constraints induced by the constraints on timelines in C_T and the assignment of V are satisfied.*

It is important to note that the CNT framework is not included in the second kind of approach mentioned in the introduction. Indeed, we do not consider the planning problem over a fixed horizon, since dimension variables are actual variables on which constraints can be enforced and propagated. For example, consider a CNT containing one dimension variable h of domain $d(h) = [1..\infty]$, one timeline $x = (\{0..2\}, h)$, and three constraints $x_1 = 0$, $x_h = 2$, and $\forall i \in [1..h-1], x_{i+1} - x_i \leq 1$. Constraint propagation techniques can remove value 1 from the domain of h, since if $h = 1$, then $x_h = 2$ and $x_1 = 0$ are not compatible. As $d(h)$ becomes $[2..\infty]$, constraint $x_2 - x_1 \leq 1$ must be satisfied. As $x_1 = 0$, constraint propagation can infer $d(x_2) = \{0, 1\}$, which in turn allows value 2 to be removed from $d(h)$. Therefore, constraints can be propagated in any direction and dimension variables will not necessarily be assigned first. Another useful feature of CNTs is the explicit presence of classical variables (outside timelines), which can model static features such as the choice of a download slot for a given observation. Last, CNTs can be easily extended to soft CNTs by replacing constraints by soft constraints, in order to model problems involving preferences such as the minimization of $card\{o \in [1..N_O] \mid nd_o = 0\}$ if downloading all observations is not possible.

Comparison with existing modeling frameworks and extensions

The CNT framework is a kind of dynamic CSP, except that in CNTs, the number of potential variables may be unbounded, if the domain of a dimension variable is infinite. In dynamic CSPs, variables are divided into a set of mandatory variables and a set of optional ones. Constraints are divided into a set of classical constraints and a set of activation constraints. The latter define when optional variables are active, as a function of the assignment of other mandatory or optional variables. Constraints are active only if their variables are active too. In CNTs, we do not explicitly define activation constraints: constraints are active depending on the domain of values of dimension variables. Compared to dynamic CSPs, a contribution of CNTs is actually that they explicitly identify the special role played by dimension variables in planning and scheduling problems.

Compared to approaches completely integrating planning into constraint programming (third class given in the introduction), CNTs are built directly over variables and constraints, and not over more general entities such as intervals or structural constraints. This allows CNTs to be very generic, since any kind of constraint can be defined to model particular features of a real-world problem, such as global constraints or constraints involving both variables and t-variables. The generic aspect of CNTs also holds since they were proved to cover many frameworks used to model discrete event dynamic systems (Anonymous 2008a), such as automata, synchronized products of automata, timed automata, STRIPS planning, Petri nets, resource-constrained project scheduling, or temporal constraint networks.

However, defining models directly over variables and constraints can be harder and less intuitive than with PDDL. In fact, higher level entities may be needed by modelers. These higher level entities can be easily added to the naturally extensible CNT framework, as shown below. The simultaneous presence of basic and high level entities in CNTs is not contradictory: it exactly fits the CSP approach, where basic and global constraints coexist.

Time reference of a timeline Some timelines can be of type *time*. Timelines of type time must have a domain of values included in $\mathbb{R}$ and must satisfy $\forall i \in [1..h(tl)-1], tl_i \leq tl_{i+1}$. With a timeline tl can be associated at most one timeline of type time, called the *time reference* of tl. If the time reference of tl is t, then tl_i represents the value of attribute tl at time t_i. Moreover, we assume that $h(tl) = h(t)$ and that $(t_i = t_j) \rightarrow (tl_i = tl_j)$, meaning that a timeline cannot take two different values at the same instant.

Also, it is sometimes useful to add variables tl_0 representing the initial state of a timeline. In this case, the timeline is said to be an *initialized timeline*.

Timeline evolution types An evolution type can be associated with each timeline tl whose time reference is not null. The evolution of a timeline is *piecewise constant* if the timeline represents features of the system that do not change between two steps. In this case, if the time reference of tl is t, then tl_i represents the value taken by timeline tl from time t_i to time t_{i+1}, time t_{i+1} excluded. The evolution of a timeline can also be *piecewise linear* if the timeline represents features that evolve linearly between two steps, or *discrete* if the timeline represents features that have no value between two steps.

Important intermediate variables Given a timeline tl whose time reference is t and given a variable x whose domain is included in $\mathbb{R}$, we can define intermediate variable $val(tl, x)$ to represent the value of timeline tl at time x. If tl has a piecewise constant evolution, then $val(tl, x) = tl_j$ with $j = \max(j \in [1..h(tl)] \mid t_j \leq x)$ if this quantity exists, $val(tl, x)$ undefined otherwise. This definition is illustrated by Figure 1. Similarly, we can define intermediate variable $valb(tl, x)$ to represent the value of timeline tl just before time x (x excluded). If tl has a piecewise constant evolution, then $valb(tl, x) = tl_j$ with $j = \max(j \in [1..h(tl)] \mid t_j < x)$ if this quantity exists, $valb(tl, x)$ undefined otherwise. For timelines whose evolution is piecewise linear or discrete, $val(tl, x)$ and $valb(tl, x)$ are defined differently.

All these variables can be handled automatically in an efficient way, with dedicated global constraints on timelines hidden to the modeler, so that (s)he can directly use quantities $val(tl, x)$ or $valb(tl, x)$ to express constraints.

Other modeling examples

Before defining algorithms, let us show the modeling capabilities of CNTs on some problems from International Planning Competitions (IPCs). For each problem, different models can be defined. As in CSPs, finding a good model may not be straightforward.

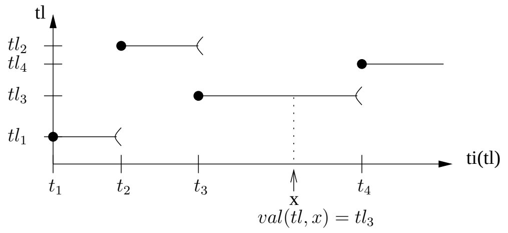

Figure 1: Semantics of a piecewise constant timeline and illustration of intermediate variables $val(tl, x)$.

Domain *BlocksWorld* (IPC2) In order to model a BlocksWorld problem, we use one dimension variable $h \in$ [*hmin*..*hmax*] representing the makespan. *hmin* is a lower bound on the number of blocks which must be moved in any valid plan, and *hmax* $= 2N$ where N is the number of blocks. We use three timelines, *move* $= ([1..N], h)$, *from* $= ([0..N], h)$, and *to* $= ([0..N], h)$, to represent respectively the block moved at step i, the block from which it is taken (0 if the block is taken from the table), and the block on which it is put on (0 if the block is put on the table). For every block b, we use timelines *top*$_b = ([0..N], h)$ and *ontable*$_b = (\{0,1\}, h)$ to represent the state of b at each step. The constraints introduced define the initial and goal states, preconditions on decisions, effects of actions, and conditions of changes of *top*$_b$ and *ontable*$_b$. But unlike with approaches like CPlan, the horizon h is not fixed initially. The expression of constraints is compact thanks to the variable-based decomposition of actions into timelines *move*, *from*, and *to*. This differs from the action-based view of STRIPS or PDDL. We also use a CSP constraint known as a *global cardinality constraint* to impose that each block can be moved at most twice (any plan in which a block is moved more than twice is not optimal). Last, we use constraints forcing some necessary moves to be done, and constraints breaking symmetric solutions.

Domain *Satellite* (IPC3) A set of N_S satellites must take a set of N_I images. Each image im has a direction $DI(im)$ and an associated image mode. Each satellite s has a set of observation instruments denoted IN_s. It is possible to compute a predicate $SUPPORTS(im, in)$ which holds if image im can be performed with instrument in. The duration needed to take a picture in a direction di' with instrument in', starting from a direction di with instrument in calibrated, is denoted $DU(di, in, di', in')$. This duration takes into account the necessity to calibrate in' if $in' \neq in$. We denote by $DUmin$ the minimum value of function $DU(\,)$. Initially, each satellite s points to a direction DII_s and instrument INI_s is calibrated. We make some simplifications here compared to the IPC3 formulation; *in the experiments, we do not make such simplifications.*

To model these problems, we use one dimension variable h_s of domain $[0..N_I]$ per satellite s. h_s represents the number of images taken by s. Timelines $im_s = ([1..N_I], h_s)$, $in_s = (IN_s, h_s)$, $di_s = (\{DI(im), im \in [1..N_I]\} \cup \{DII_s\}, h_s)$, and $t_s = ([0..Tmax], h_s)$ represent respectively images taken by s, associated instruments, associated directions, and the times when images are finished (it is possible to compute a maximum time $Tmax$ for the domain of t_s). Timelines in_s, dir_s, t_s are initialized, so that variables $in_{s,0}$, $dir_{s,0}$, $t_{s,0}$ can be used. A variable denoted $tend$ represents the total duration needed to take all pictures. Different constraints are defined. Constraints c_9 and c_{10} ensure that each image is taken exactly once. This enables suboptimal solutions to be pruned. Constraint c_{11} defines the initial state. Constraint c_{12} imposes feasibility constraints on the decisions. Constraints c_{13} and c_{14} describe the evolution of directions and times. Constraint c_{15} is redundant but crucial for the algorithmic efficiency. Constraint c_{16} defines $tend$ as the makespan, which must be minimized.

$$\sum_{s\in[1..N_S]} h_s = N_I \qquad (c_9)$$
$$alldifferent(\{im_{s,i} \mid s\in[1..N_S], i\in[1..h_s]\}) \qquad (c_{10})$$
$$\forall s\in[1..N_S],\quad (di_{s,0}=DII_s)\wedge(in_{s,0}=INI_s)\wedge(t_{s,0}=0) \qquad (c_{11})$$
$$\forall s\in[1..N_S], \forall i\in[1..h_s],$$
$$SUPPORTS(im_{s,i}, in_{s,i}) \qquad (c_{12})$$
$$di_{s,i} = DI(im_{s,i}) \qquad (c_{13})$$
$$t_{s,i} = t_{s,i-1} + DU(di_{s,i-1}, in_{s,i-1}, di_{s,i}, in_{s,i}) \qquad (c_{14})$$
$$tend \geq t_{s,i} + (h_s - i)\cdot DUmin \qquad (c_{15})$$
$$tend = \max_{s\in[1..N_S]} t_{s,h_s} \qquad (c_{16})$$

Domain *Trucks* (IPC5) The modeling of this domain shows how useful intermediate variables $val(tl,x)$ and $valb(tl,x)$ are. Domain Trucks involves a set of packages P and a set of trucks T. Each truck $\tau\in T$, initially located at a location denoted LI_τ, has a limited capacity and can load packages, unload packages, and drive between locations. Each package $p\in P$ must be transfered from an initial location LI_p to a goal location LG_p. Actions have a given duration (the duration of an unloading is denoted DU), and the goal is to minimize the makespan. In the model given here, *but not in the experiments*, we make simplifications compared to the IPC5 version and we omit quantification on τ and p in the expressions of constraints.

For each truck τ, we use one dimension variable h_τ representing the number of actions performed by τ, and a set of timelines $\{t_\tau, a_\tau, p_\tau, l_\tau, n_\tau\}$ of dimension h_τ. For each $i\in[1..h_\tau]$, $a_{\tau,i}$ represents the action made by τ at step i (load, unload, or drive), $p_{\tau,i}$ represents the package concerned by the action (if any), $t_{\tau,i}$ is the start time of the action, and $l_{\tau,i}$ and $n_{\tau,i}$ are the location of τ and the number of packages in τ at the end of the action. Timeline t_τ is the time reference of timelines a_τ and p_τ. For each package p, we use one dimension variable h_p and two timelines $\{t_p, l_p\}$ of dimension h_p. For each step $j\in[1..h_p]$, $l_{p,j}$ represents the location of p at time $t_{p,j}$. We consider that the location $l_{p,j}$ can also be a truck. Timeline t_p is the time reference of l_p. All timelines have a discrete evolution except from l_p, whose evolution is considered to be piecewise constant. Timelines t_τ, l_τ, t_p, l_p are initialized.

Constraints c_{17} to c_{27} are imposed over these timelines. For example, c_{21} defines the evolution of the number of packages in a truck. If a package is loaded by a truck, it must be at the same location as the truck just before the start of the loading (c_{22}). If a package is unloaded by a truck, it is at the same location as the truck at the end of the unloading (c_{23}). If a package is in a truck, then it has just been loaded and will be unloaded at the next step (c_{24}). In fact, from the start of the loading to just before the end of the unloading, the package is considered to be in the truck. c_{26} defines the makespan, and c_{27} asserts both that a package must be at its goal location at the end and that when it is at its goal location, then its associated timeline is over.

$$(t_{\tau,0}=t_{p,0}=0)\wedge(l_{\tau,0}=LI_\tau)\wedge(l_{p,0}=LI_p) \qquad (c_{17})$$
$$t_{\tau,i}\geq t_{\tau,i-1}+duration(a_{\tau,i}, l_{\tau,i-1}, l_{\tau,i}) \qquad (c_{18})$$
$$(p_{\tau,i}=0)\leftrightarrow(a_{\tau,i}=drive) \qquad (c_{19})$$
$$(l_{\tau,i}\neq l_{\tau,i-1})\leftrightarrow(a_{\tau,i}=drive) \qquad (c_{20})$$
$$n_{\tau,i}=n_{\tau,i-1}+f(a_{\tau,i}) \qquad (c_{21})$$
$$\text{with } f(load)=1, f(unload)=-1, f(drive)=0$$
$$(a_{\tau,i}=load)\rightarrow\ ((valb(l_{p_{\tau,i}}, t_{\tau,i})=l_{\tau,i-1}) \wedge(val(l_{p_{\tau,i}}, t_{\tau,i})=\tau)) \qquad (c_{22})$$
$$(a_{\tau,i}=unload)\rightarrow\ ((valb(l_{p_{\tau,i}}, t_{\tau,i}+DU)=\tau) \wedge(val(l_{p_{\tau,i}}, t_{\tau,i}+DU)=l_{\tau,i})) \qquad (c_{23})$$
$$(l_{p,j}=\tau)\rightarrow\ ((val(a_\tau, t_{p,j})=load) \wedge(val(p_\tau, t_{p,j})=p) \wedge(val(a_\tau, t_{p,j+1}-DU)=unload) \wedge(val(a_\tau, t_{p,j+1}-DU)=p)) \qquad (c_{24})$$
$$(l_{p,j}\in T)\leftrightarrow(l_{p,j+1}\notin T) \qquad (c_{25})$$
$$tend=\max_{p\in[1..N_P]} t_{p,h_p} \qquad (c_{26})$$
$$(l_{p,j}=LG_p)\leftrightarrow(j=h_p) \qquad (c_{27})$$

Other constraints are added to get a more efficient model. For example, c_{28} prevents a package from being at the same place at two different steps, and c_{29} is a transition constraint pruning suboptimal decisions from the search space.

$$alldifferent(l_{p,j} \mid j\in[0..h_p]) \qquad (c_{28})$$
$$(a_{\tau,i}=load)\rightarrow(a_{\tau,i+1}\neq unload) \qquad (c_{29})$$

A dynamic depth-first tree search using constraint propagation for CNTs

The algorithm presented is a standard depth-first tree search using constraint propagation, enhanced with an extension phase that dynamically changes the problem by adding new variables and constraints whenever the minimum value in the domain of a dimension variable is modified. This extension phase is combined with constraint propagation, which can generate value removals, which can in turn trigger a new extension phase, and so on until a fixed point is reached. The interest of constraint propagation is to simplify the current problem by pruning inconsistent values or tuples of values. The algorithm is called *dynDFS*. It is directly defined in an optimization context, that is we assume that there exists an objective variable denoted obj whose value must be minimized. If the algorithm terminates, it returns an optimal CNT assignment if there exists one, and null otherwise. The main steps of the algorithm are detailed below.

Function *dynPropagate* During search, a current CSP (V, C_V) is maintained (V is the set of variables and C_V the set of constraints). The iterative extension and propagation phases are performed by function *dynPropagate*. While constraints need to be propagated and while the current CSP is not known to be inconsistent (line 31), constraints are propagated with function *propagate* (line 33).

(R1) *propagate*(V, C_V) transforms the CSP (V, C_V) into an equivalent CSP (V', C'_V),[1] by enforcing at least *backward checking*. This means that in (V', C'_V), each constraint whose scope is completely instantiated is satisfied.

To satisfy requirement (R1), *propagate* can be any standard constraint propagation scheme, such as forward checking, arc consistency, or path consistency (Dechter 2003).

The CSP obtained after constraint propagation can then be extended by calling *extend*(V', C'_V, T, C_T, A_H) (line 34). In this call, A_H corresponds to the previous minimal assignment of the dimension variables.

(R2) *extend*(V, C_V, T, C_T, A_H) returns a triple (V', C'_V, b) such that

- the CNTs (V, C_V, T, C_T) and (V', C'_V, T, C_T) are equivalent[1],
- b equals true iff $(V', C'_V) \neq (V, C_V)$,
- for every constraint $(S_V, S_T, fct) \in C_T$ such that there is a unique possible assignment A for the dimension variables of the timelines in S_T, C'_V contains $fct(A)$.

Requirement (R2) can be fulfilled in different ways. The laziest version of *extend* consists in generating constraints only when all dimension variables are assigned. The approach we use in the experiments is still lazy, but more incremental: when *extend*(V', C'_V, T, C_T, A_H) is called, it is possible to compare A_H, the previous minimum assignment of the dimension variables, and A'_H, the current minimum assignment of the dimension variables, and to add the set of t-variables $\{tl_i \mid tl \in T, i \in [A_H[h(tl)] + 1..A'_H[h(tl)]]$ to V'. The way constraints are added to C'_V depends on the type of constraint considered. For example,

- for a constraint such as $\forall i \in [1..h(tl)], tl_i \neq x$, function *extend* can add the set of constraints $\{tl_i \neq x \mid i \in [A_H[h(tl)] + 1..A'_H[h(tl)]]$ to C'_V;
- a constraint like *alldifferent*$(tl_i \mid i \in [1..h(tl)])$ can generate constraint *alldifferent*$(tl_i \mid i \in [1..A'_H[h(tl)]])$ if $A_H[h(tl)] \neq A'_H[h(tl)]$.

In fact, constraints can be added as soon as they must necessarily be satisfied. The design of specialized schemes for function *extend* for a constraint on timelines (S_V, S_T, fct) can be highly dependent on *fct* and is not discussed here.

Functions *dynDFS* and *recDynDFS* Given a CNT (V, C_V, T, C_T), the systematic depth-first tree search is performed by calling *dynDFS*(V, C_V, T, C_T). After an initial extension/propagation step (lines 4 and 5), function *dynDFS* calls function *recDynDFS* if the initial problem has not been proved to be inconsistent, and returns *null* otherwise.

[1]That is, they define the same set of consistent assignments.

Algorithm 1: dynDFS, a dynamic depth-first tree search using constraint programming.

```
1  dynDFS($V, C_V, T, C_T$)
2  begin
3      $A_H \leftarrow \{(h,0) \mid h \in H\}$
4      $(V, C_V) \leftarrow$ extend$(V, C_V, T, C_T, A_H)$
5      $(V, C_V) \leftarrow$ dynPropagate$(V, C_V, T, C_T)$
6      if $\forall x \in V, d(x) \neq \varnothing$ then
7          return recDynDFS$(V, C_V, T, C_T)$
8      else return $null$
   end

10 recDynDFS($V, C_V, T, C_T$)
11 begin
12     if $\forall x \in V, card(d(x)) = 1$ then
13         return $\{(x,a) \mid x \in V, a \in d(x)\}$
14     else
15         Choose $x \in V$ s.t. $card(d(x)) > 1$
16         Choose a partition $\{D_1, D_2\}$ of $d(x)$
17         $(A, opt) \leftarrow (null, +\infty)$
18         foreach $k \in \{1,2\}$ do
19             $(V', C'_V) \leftarrow (V, C_V \cup \{obj < opt\})$
20             $d'(x) \leftarrow D_k$
21             $(V', C'_V) \leftarrow$ dynPropagate$(V', C'_V, T, C_T)$
22             if $\forall x \in V', d'(x) \neq \varnothing$ then
23                 $A' \leftarrow$ recDynDFS$(V', C'_V, T, C_T)$
24                 if $A' \neq null$ then $(A, opt) \leftarrow (A', A'[obj])$
25         return $A$
   end

27 dynPropagate($V, C_V, T, C_T$)
28 begin
29     $(V', C'_V) \leftarrow (V, C_V)$
30     $b \leftarrow true$
31     while $b \wedge (\forall x \in V' \mid d'(x) \neq \varnothing)$ do
32         $A_H \leftarrow \{(h(tl), min(d'(h(tl))) \mid tl \in T\}$
33         $(V', C'_V) \leftarrow$ propagate$(V', C'_V)$
34         $(V', C'_V, b) \leftarrow$ extend$(V', C'_V, T, C_T, A_H)$
35     return $(V', C'_V)$
   end
```

If it terminates, *recDynDFS*(V, C_V, T, C_T) returns an optimal assignment A of the CNT (V, C_V, T, C_T) if it exists, and *null* otherwise. If there is a unique possible assignment of V, this assignment is returned (lines 12-13). Otherwise, the algorithm chooses a variable $x \in V$ not assigned yet and builds a partition of the domain of x, according to some heuristics (lines 15-16). The two search subspaces defined by this partition are then successively explored (lines 18 to 24). For each of them, *recDynDFS* first propagates constraints using *dynPropagate* (line 21). If no inconsistency is revealed (line 22), *recDynDFS* is recursively called (line 23), If a solution $A' \neq$ *null* is returned, it is recorded as well as the best value known for the objective.

Discussion and properties Algorithm *dynDFS* is a generic algorithm which covers several existing approaches. Indeed, approaches reasoning over a sequence of size-bounded CSPs simply correspond to variable/value choice heuristics (lines 15 and 16) where all dimension variables are assigned first, with their minimal values. *dynDFS* can also adopt a strategy where horizons are dynamically incremented during search, when constraint propagation prunes the minimum value in the domain of dimension variables. As a result, *dynDFS* allows several approaches to be compared inside a common framework.

Formal properties of *dynDFS* are given below. This algorithm is correct but does not necessarily terminate, since it might get trapped in infinite branches of the search space when the domain of some variable is infinite.

Proposition 1 *(Correction) If function* propagate *satisfies (R1) and function* extend *satisfies (R2), then* dynDFS *is correct: if it returns a result, this result is an optimal assignment if the CNT considered admits a solution and* null *otherwise.*

Proposition 2 *(Termination) If all domains of values are finite, then* dynDFS *terminates. If all non-dimension variables have a finite domain and if the problem admits as least one solution, then there exist choice heuristics (lines 15 and 16) such that* dynDFS *finds a consistent assignment in a finite time. In general,* dynDFS *does not terminate.*

Proposition 3 *(Complexity class of CNTs) Deciding whether there exists a CNT assignment with an objective value lesser than a given threshold θ is (a) NP-complete for CNTs where all domains of values are finite, (b) semi-decidable for CNTs such that all non-dimension variables have a finite domain, and (c) undecidable in general.*

Sketch of the proofs: for Prop. 1, the idea is to prove that if it terminates, dynPropagate(V, C_V, T, C_T) *returns a couple (V', C'_V) such that (V, C_V, T, C_T) and (V', C'_V, T, C_T) are equivalent, and that if it terminates,* recDynDFS(V, C_V, T, C_T) *returns an optimal assignment of (V, C_V, T, C_T) if there exists one and null otherwise. For Prop. 2, if all domains of non-dimension variables are finite, it suffices to use an assignment heuristics that iteratively increments the maximum value that can be assigned to a dimension variable. For Prop. 3, checking the consistency of a CNT assignment is polynomial and any finite CSP can be expressed as a CNT, hence the NP-completeness result; if all non-dimension variables have a finite domain, then Prop. 2 implies the semi-decidability result; for undecidability in general, it was shown in (Anonymous 2008a) that the halting problem can be expressed as the problem of finding a consistent assignment of a CNT.*

Experiments

To measure the practical interest of CNTs, we performed experiments on domains BlocksWorld (IPC2), Satellite (*propositional* and *simpletime* versions, IPC3), and Trucks (*propositional* and *temporal* versions, IPC5). The first task was to build CNT representations manually as described previously. For these domains, the value of dimension variables can be bounded while preserving optimality, hence *dynDFS* terminates. The goal is to minimize the makespan.

The ideas of CNT and *dynDFS* are implemented over Choco (Laburthe & the OCRE project team 2000), a constraint programming library. The constraint propagation algorithm used is GAC (Generalized Arc Consistency (Dechter 2003)). Function *extend* is simulated via constraints *ifThen*$(h \geq i, c_i)$, which activate constraint c_i only when guard $h \geq i$ holds. Several parameter settings were tested for the choice of the variable to consider at each step: (1) consider dimension variables first; (2) consider non-dimension variables first; (3) consider dimension and non-dimension variables in any order. The results presented for BlocksWorld and Satellite are obtained with option (3). As a secondary criterion, we use the standard CSP heuristics that chooses a variable of minimum domain size. The results presented for Trucks correspond to option (2), and by considering first variables having a minimum domain size for non-dimension variables, and variables having a minimum minimal value for dimension variables.

We ran our experiments on an AMD Opteron processor, 2.4 GHz, with 1GB RAM, under Linux, with a time limit of half an hour per problem. We compared *dynDFS*(CNT) with the optimal planners awarded at the last planning competition: MaxPlan, SatPlan, and CPT.[2] MaxPlan and SatPlan can handle propositional domains. CPT can handle both propositional and temporal domains.

Table 1 shows that in general, *dynDFS*(CNT) performs better than MaxPlan, SatPlan, and CPT. On small-size instances, *dynDFS*(CNT) can be slower since, as it contains more information, the initialization can be longer. On harder instances, *dynDFS*(CNT) provides significant gains. Instances of BlocksWorld are easy for *dynDFS*(CNT) thanks to symmetry breaking constraints and to constraints forcing necessary moves to be done. Instances of Satellite, propositional or temporal, are solved in a few seconds with *dynDFS*(CNT), whereas with MaxPlan, SatPlan, and CPT, which work on models containing less information, they are solved only in several minutes or unsolved at all. Trucks appears to be more challenging, in the sense that the CNT representation speeds search, but does not modify the intrinsic complexity of the problem.

For unsolved instances, as shown in Figure 2, *dynDFS*(CNT) is able to quickly produce solutions whose quality is better than the quality of the solution produced by SGPlan, a heuristic-based planner awarded in IPC5. In fact, for domain Trucks, *dynDFS*(CNT) is quite anytime: the optimal solution is reached quite quickly, and the rest of the time is dedicated to prove optimality,

Conclusion

In this paper, we presented Constraint Networks on Timelines (CNTs), a generic constraint-based framework for modeling and solving planning and scheduling problems. This framework is compact and has a clear semantics based on variables and constraints. A generic dynamic depth-first tree search algorithm using constraint propagation has been developed and tested on several instances taken from planning competitions. Experimental results have shown the

[2]For MaxPlan, see http://www.cse.wustl.edu/∼chen/maxplan/. For SatPlan, see http://www.cs.rochester.edu/∼kautz/satplan. For CPT, see http://www.cril.univ-artois.fr/∼vidal/#cpt. For CPT, we use CPT1 because CPT2 is not publicly available.

	CPU time (sec.)				Makespan
	MaxPlan	SatPlan	CPT	dynDFS(CNT)	
bw-large-a	0.51	0.38	0.14	1.08	(12)
bw-large-b	4.64	2.36	0.96	2.60	(18)
bw-large-c	171.19	38.99	56.17	6.94	(28)
bw-large-d	-	455.65	-	15.14	(36)
bw-ipc10	0.47	0.24	0.03	0.79	(20)
bw-ipc20	-	5.42	407.20	1.48	(32)
bw-ipc30	-	44.28	-	2.55	(36)
bw-ipc40	-	183.91	151.33	4.92	(58)
bw-ipc50	-	-	-	8.39	(86)
satellite05-prop	0.56	0.43	0.58	0.52	(7)
satellite06-prop	0.32	0.47	0.16	0.54	(8)
satellite07-prop	0.46	0.62	0.40	0.58	(6)
satellite08-prop	25.45	40.73	19.24	1.53	(8)
satellite09-prop	1.76	2.77	1.61	0.89	(6)
satellite10-prop	54.28	16.58	40.83	0.98	(8)
satellite11-prop	7.87	12.28	4.40	0.88	(8)
satellite12-prop	536.40	317.72	-	8.85	(14)
satellite13-prop	644.82	308.38	-	10.04	(13)
satellite14-prop	95.80	74.37	-	8.27	(8)
trucks01-prop	0.71	0.53	11.65	1.29	(11)
trucks02-prop	4.49	2.46	-	4.37	(14)
trucks03-prop	30.95	60.74	-	22.66	(16)
trucks04-prop	1163.32	409.21	-	15.61	(18)
trucks05-prop	-	-	-	45.97	(19)
trucks06-prop	-	-	-	463.41	(22)
trucks07-prop	709.74	642.51	-	1780.32	(18)
satellite05-simpletime			1.21	0.51	(36)
satellite06-simpletime			0.26	0.42	(46)
satellite07-simpletime			0.23	0.54	(34)
satellite08-simpletime			1329.60	0.97	(46)
satellite09-simpletime			3.63	0.87	(34)
satellite10-simpletime			875.18	4.50	(43)
satellite11-simpletime			26.82	1.08	(46)
satellite12-simpletime			-	14.01	(79)
trucks01-temporal				2.38	(843.2)
trucks02-temporal			-	6.35	(1711.4)
trucks03-temporal			-	14.69	(1202.6)
trucks04-temporal			-	-	( )
trucks05-temporal			-	-	( )
trucks06-temporal			-	-	( )

Table 1: Comparison between *dynDFS*(CNT) and some optimal planners, on propositional and temporal domains.

practical interest of the approach, both compared to existing optimal planners in terms of time to get the optimal solution and to prove optimality, and compared to heuristic planners in terms of solution quality. In particular, some problems unsolved by existing optimal planners are solved in a few seconds with CNTs. In the end, the gains in using CNTs are twofold. From a modeling point of view, the basic constraint-based semantics allows various kinds of information to be captured in CNTs, such as constraints modeling scheduling aspects as well as planning aspects, temporal constraints, constraints on both dimension and timeline variables, or constraints on binary or n-ary variables. From an algorithmic point of view, CNTs allow efficient models containing information such as global constraints, constraints between states, constraints between actions, symmetry breaking constraints, constraints pruning suboptimal solutions, or redundant constraints, to be developed. Exploiting the information available avoids the planner from being blind, while preserving optimality.

In the future, we believe that the performance of algorithms on CNTs could be improved significantly, since constraint programming techniques such as intelligent backtracking, structural decomposition, improved value choice heuristics, limited discrepancy search, soft constraint propagation, constraint preprocessing, or randomization and restart, have not been used yet. It would also be interesting to develop approximate algorithms and to compare their performance with heuristic-based planners. Last, the approach should be extended in order to be able to handle uncertainty or to build flexible plans.

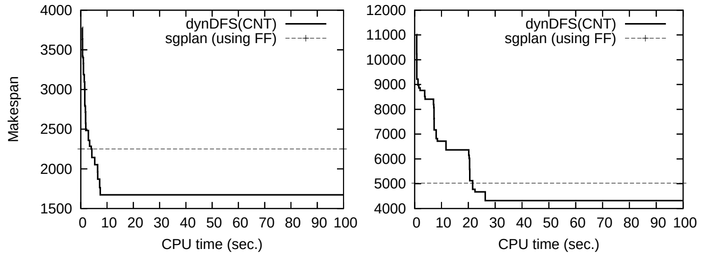

Figure 2: Evolution of the makespan given by *dynDFS*(CNT) and comparison with the makespan given by an heuristic-based planner (SGPlan, which uses FF), on problems trucks05-temporal (left) and trucks06-temporal (right).

References

[Anonymous 2008a] Reference withheld for blind review.

[Anonymous 2008b] Reference withheld for blind review.

[Blum & Furst 1997] Blum, A., and Furst, M. 1997. Fast Planning through Plan Graph Analysis. *Artificial Intelligence* 90:281–300.

[Dechter 2003] Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

[Do & Kambhampati 2001] Do, M., and Kambhampati, S. 2001. Planning as Constraint Satisfaction: Solving the Planning-Graph by Compiling it into CSP. *Artificial Intelligence* 132(2):151–182.

[Fikes & Nilsson 1971] Fikes, R., and Nilsson, N. 1971. STRIPS: a New Approach to the Application of Theorem Proving. *Artificial Intelligence* 2(3-4):189–208.

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

[Kautz & Selman 1992] Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *Proc. of ECAI-92*, 359–363.

[Laburthe & the OCRE project team 2000] Laburthe, F. 2000. CHOCO: Implementing a CP Kernel. In *Proc. of CP-00*.

[Lopez & Bacchus 2003] Lopez, A., and Bacchus, F. 2003. Generalizing GraphPlan by Formulating Planning as a CSP. In *Proc. of IJCAI-03*, 954–960.

[McDermott 1998] McDermott, D. 1998. PDDL, the Planning Domain Definition Language. Technical report.

[Mittal & Falkenhainer 1990] Mittal, S., and Falkenhainer, B. 1990. Dynamic Constraint Satisfaction Problems. In *Proc. of AAAI-90*, 25–32.

[Nareyek *et al.* 2005] Nareyek, A.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H.; Rintanen, J.; and Tate, A. 2005. Constraints and AI Planning. *IEEE Intelligent Systems* 20(2):62–72.

[Nareyek 2001] Nareyek, A. 2001. *Constraint-Based Agents*. Springer.

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

[Vidal & Geffner 2006] Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence* 170:298–335.