Programmation par contraintes sur les séquences infinies

Xavier Dupont, Arnaud Lallouet, Y.C. Law, J.H.M. Lee, C.F.K. Siu

Programmation par Contraintes sur les Séquences Infinies

Xavier Dupont[1] **Arnaud Lallouet**[1]

Y. C. Law[2] **J. H. M. Lee**[2] **C. F. K. Siu**[2]

[1]Université de Caen Basse-Normandie, 14000, Caen
[2]Université Chinoise de Hong-Kong
{xavier.dupont,arnaud.lallouet}@unicaen.fr
{yclaw,jlee,fksiu}@cse.cuhk.edu.hk

Abstract

La programmation par contraintes est habituellement utilisée pour résoudre des problèmes combinatoires pour lesquels l'ensemble des solutions est de cardinalité finie. Cette approche, bien adaptée aux problèmes d'ordonnancement, est moins applicable aux problèmes de planification, pour lesquels l'horizon doit être borné, et est complètement inapplicable à des problèmes temporels sans horizon. Nous proposons ici un cadre alternatif dans lequel les variables sont des séquences symboliques infinies. Il devient ainsi possible de spécifier des contraintes sur ces variables et de générer des solutions qui satisfont ces contraintes. Bien que chaque solution prise indépendamment soit de taille infinie, et que le nombre de solutions lui-même soit en général infini, il est possible de représenter l'ensemble des solutions de telle sorte qu'une solution arbitraire puisse être générée de façon efficace.

1 Introduction

La programmation par contraintes est un paradigme de programmation qui permet de résoudre des problèmes combinatoires de grandes tailles d'une manière générique. Pour résoudre un problème de programmation par contraintes, l'utilisateur spécifie les variables et les contraintes du problème et laisse le système trouver des solutions, c'est-à-dire des affectations de valeurs aux variables qui satisfont toutes les contraintes.

Cette approche est utilisée pour résoudre des problèmes d'allocation de ressources, de vérification de circuits électroniques, ou de raisonnement spatial et temporel. Certains types de problèmes, comme les problèmes d'ordonnancement, correspondent à des disciplines spécialisées de la programmation par contraintes, pour lesquelles des algorithmes de propagation spécifiques ont été développés [2]. En raison des succès de cette approche, la tendance est maintenant à l'extension du champ d'application de la programmation par contraintes à d'autres domaines, qui n'étaient pas habituellement couverts, comme la fouille de données [6] ou les logiques temporelles, qui sont le sujet de cet article.

Malgré la diversité des domaines d'application, il reste difficile de modéliser des problèmes dont la valeur des variables varie au cours du temps. Les solutions sont toujours des affectations de valeurs à un ensemble fini de variables, ce qui ne permet pas de modéliser l'évolution des valeurs d'une variable sur une durée arbitraire non définie à l'avance.

Ces caractéristiques ne posent pas beaucoup de problèmes pour la modélisation de problèmes d'ordonnancement, où l'objectif est en général de minimiser le temps nécessaire pour accomplir un ensemble de tâches dont une borne supérieure sur le temps nécessaire est connue à l'avance. La situation est plus délicate pour les problèmes de planification, pour lesquels le but est de minimiser le temps nécessaire pour atteindre une configuration désirable, et ce sans aucune information connue a priori concernant ce temps. Des méthodes spécifiques ont été élaborées pour la résolution de ce type de problème, et les approches basées sur la programmation par contraintes utilisent en général une résolution itérative, qui consiste à allonger le temps imparti jusqu'à ce qu'une solution soit trouvée [5]. Les ressources nécessaires à la résolution augmentent au

fur et à mesure des résolutions infructueuses, ce qui rend parfois l'approche inapplicable.

Enfin, les domaines de variables existants ne permettent pas de modéliser des problèmes temporels sans horizon, pour lesquels l'objectif est simplement de satisfaire des contraintes temporelles, et ce de façon indéfinie sur la durée. Une solution sera alors une séquence infinie d'affectations, qui devra à tout instant pouvoir être étendue de façon à continuer de satisfaire toutes les contraintes.

Des formalismes, parfois très généraux [9], ont été développés pour modéliser ce type de problème. Mais ces approches fonctionnent toutes par extension de l'horizon temporel, et font penser aux méthodes utilisées pour la résolution de problèmes de planification avec des CSPs.

Nous présentons dans cet article un formalisme pour traiter des problèmes temporels en utilisant les techniques de la programmation par contraintes. Les applications couvrent des domaines qui étaient jusqu'à présent exclus du domaine pour les raisons évoquées. Des exemples sont la vérification de systèmes interactifs ou dynamiques, la planification sans horizon, et l'ajustement optimal de systèmes réactifs à un environnement changeant. Des applications particulières peuvent être l'ajustement en continu d'une chaîne de production à certaines conditions, telles que la disponibilité des produits, ou bien, dans un domaine plus ludique, la génération en continue de musique à partir d'un ensemble de règles d'harmonisation.

2 Problème de satisfaction de contraintes

Un problème de satisfaction de contraintes (CSP), au sens large, est un tuple $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ où $\mathcal{X}$ est un ensemble fini de variables, $\mathcal{D}$ est l'ensemble des domaines associés aux variables, et $\mathcal{C}$ est un ensemble fini de contraintes. Pour chaque variable $X \in \mathcal{X}$, on note $D(X) \in \mathcal{D}$ le domaine associé à X.

Une contrainte $C \in \mathcal{C}$ peut être vue comme l'application d'une relation R d'arité a, notée $R(C)$, sur un tuple $\tau = (\tau[1], \tau[2], \ldots, \tau[a])$ de a variables, appelé portée de C et notée $\pi(C)$, où R est un sous-ensemble du produit cartésien des domaines des variables de τ, c'est-à-dire $R \subseteq D(\tau)$ avec $D(\tau) = D(\tau[1]) \times D(\tau[2]) \times \cdots \times D(\tau[a])$. La contrainte est satisfaite par une affectation $\alpha \in D(\tau)$ si $\alpha \in R$.

Une solution s d'un CSP dont les variables sont $\mathcal{X} = (X_1, X_2, \ldots, X_n)$ est l'affectation d'une valeur à chaque variable, où chaque valeur est prise dans le domaine de la variable correspondante, et de telle sorte que toutes les contraintes sont satisfaites, c'est-à-dire que $s \in D(X_1) \times D(X_2) \times \cdots \times D(X_n)$ et

$$\forall C \in \mathcal{C}, s_{|\pi(C)} \in R(C)$$

où pour une contrainte C et une affectation s, $s_{|\pi(C)}$ dénote la projection de s sur la portée de C, c'est-à-dire l'affectation partielle qui à chaque variable de la portée de C associe la valeur correspondante de s.

L'objectif est de trouver la ou les solutions qui satisfont toutes les contraintes.

En général, les domaines des variables sont de cardinalités finies, ce qui a pour conséquence que le nombre de solutions possibles et la taille de l'espace de recherche sont finis. Dans ces conditions, l'espace de recherche peut toujours être parcouru intégralement. L'objectif est donc de rendre la recherche de solutions aussi rapide que possible en identifiant des régions infructueuses dont on pourra éviter l'exploration. Cela se fait à l'aide de propagateurs qui permettent d'exploiter la sémantique des contraintes afin de réduire le domaine des variables et choisir des valeurs intéressantes pour la suite de la recherche.

Ces caractéristiques des CSPs ne sont pas présentes dans les StCSPs. En effet, l'espace de recherche n'est pas de taille finie, et le nombre de solutions possibles est en général infini. Cependant, pour certains types de contraintes, il est possible de représenter l'ensemble des solutions par une structure de données de taille finie, qui permet de générer n'importe quelle solution de façon efficace, et cette structure peut être calculée par résolution d'un CSP.

3 StCSPs

Dans un problème de Satisfaction de Contraintes sur Flux (StCSP), de l'anglais "Stream Constraint Satisfaction Problem", les domaines des variables sont des ensembles de flux, et les contraintes sont appliquées sur des termes. Comme la notion de contrainte est trop générale, nous considérons seulement des contraintes dites "de voisinage". Ces notions sont détaillées dans ce qui suit.

3.1 Flux

Les StCSPs sont des CSPs dont les domaines des variables sont des ensembles de flux. Un *flux* α est une séquence ordonnée $\langle \alpha(0), \alpha(1), \ldots \rangle$ de datons, où chaque daton $\alpha(i)$ appartient à un ensemble fini commun Σ_α, l'alphabet associé à α. On note $\alpha(i, j)$, $0 \leq i \leq j$ la séquence finie $\langle \alpha(i), \alpha(i+1), \ldots, \alpha(j) \rangle$. $\alpha(i, \infty)$ correspond à la séquence infinie $\langle \alpha(i), \alpha(i+1), \ldots \rangle$, et $\alpha(i, i)$ correspond à $\langle \alpha(i) \rangle$, qui est identifiable à $\alpha(i)$.

Nous définissons des opérateurs sur les flux qui permettent de symboliser des parties d'un flux.

L'opérateur "Next" associe à un flux le même flux amputé de son premier daton. Pour un flux $\alpha = \alpha(0, \infty)$, $\text{Next}(\alpha) = \alpha(1, \infty)$.

L'opérateur "Next" doit être considéré comme un opérateur de projection. Le flux renvoyé par "Next" est intrinsèquement le même que le flux d'entrée. En particulier, les datons oubliés par "Next" restent définis. L'opérateur "Next" sert donc à calculer différentes vues d'un même flux.

Cet opérateur peut être composé. Nous notons $\text{Next}^n(\alpha)$ le flux $\text{Next}(\text{Next}^{n-1}(\alpha))$, avec $\text{Next}^0(\alpha) = \alpha$.

Les flux peuvent être représentés à l'aide de l'opérateur $(\cdot)^\omega$. On note a^ω le flux $\langle a, a, \ldots \rangle$, d'alphabet $\Sigma = \{a\}$. Cette notation peut être étendue aux mots finis. Si $w = \langle w[1], w[2], \ldots, w[\ell] \rangle$ est un mot de longueur ℓ, w^ω représente le flux $\langle w[0], w[1], \ldots, w[\ell], w[0], w[1], \ldots, w[\ell], \ldots \rangle$.

Dans le cas où S est un ensemble de mots finis, S^ω représente l'ensemble des mots infinis qui peuvent être construits avec des mots de S. Une définition récursive est $S^\omega = \{ww' \mid w \in S \wedge w' \in S^\omega\}$. En particulier, l'ensemble des mots infinis sur un alphabet fini Σ est noté Σ^ω.

Un exemple de flux est le flux $\alpha = (abc)^\omega = \langle a, b, c, a, b, c, \ldots \rangle$. Pour ce flux, $\text{Next}^2(\alpha) = \langle c, a, b, c, a, b, \ldots \rangle$ et $\text{Next}^3(\alpha) = \alpha$.

3.2 Variables flux

Cette section introduit le type de variable utilisé dans les StCSPs, que nous appelons *variable flux*. Le domaine d'une variable flux est un ensemble de flux, potentiellement infini. Le domaine initial d'une variable flux est l'ensemble des séquences qui peuvent être construites sur un alphabet fini donné. Si X est une variable flux et $d(X)$ est l'alphabet associé à X, le domaine de X est $D(X) = d(X)^\omega$.

L'opérateurs "Next" est étendu aux domaines de variables flux. Pour une variable flux X de domaine $D(X)$, $\text{Next}(D(X)) = \{\text{Next}(\alpha) \mid \alpha \in D(X)\}$.

Les variables flux sont identifiées avec leurs domaines. Pour une variable flux X, $\text{Next}(X) = \text{Next}(D(X))$.

3.3 Termes

Un *terme* est construit par application de l'opérateur "Next" sur une variable flux. L'ensemble des termes peut être construit inductivement :

- une variable flux est un terme
- si t est un terme, alors $\text{Next}(t)$ est un terme

La notion de domaine est étendue aux termes. Pour une variable flux, le domaine du terme est simplement le domaine de la variable. Sinon, t est de la forme $\text{Next}(t')$, où t' est un terme, et $D(t) = D(\text{Next}(t')) = \{\text{Next}(\alpha) \mid \alpha \in D(t')\}$.

Un terme peut toujours être écrit sous la forme $\text{Next}^n(X)$, où X est une variable flux et $n \in \mathbb{N}$. Par convention, $\text{Next}^0(X) = X$ et $\text{Next}^n(X) = \text{Next}(\text{Next}^{n-1}(X))$ si $n \neq 0$. Le domaine d'un terme est donc $D(t) = D(\text{Next}^n(X)) = \{\text{Next}^n(\alpha) \mid \alpha \in D(X)\}$.

Exemple Soit X une variable flux de domaine $D(X) = a(b)^\omega$. Alors $\text{Next}(X)$ est un terme, dont le domaine est $D(\text{Next}(X)) = b^\omega$ et $D(\text{Next}^n(X)) = D(\text{Next}(X)) = b^\omega$ pour tout n strictement positif.

3.4 Contraintes

Pour la formalisation des contraintes sur les StCSPs, nous réutilisons les notations de la section 2.

Une contrainte est l'application d'une relation R d'arité a sur un tuple τ de a termes. Pour une contrainte C de portée $\pi(C) = (t_1, t_2, \ldots, t_n)$, la relation $R(C)$ associée à C décrit un sous-ensemble du produit cartésien des domaines des variables de sa portée, c'est-à-dire $R(C) \subseteq D(t_1) \times D(t_2) \times \cdots \times D(t_n)$.

Une affectation $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in D(t_1) \times D(t_2) \times \cdots \times D(t_n)$ satisfait C si $\alpha \in R_C$.

Cette formulation est très générale. Un exemple de contrainte est $C : R_C(t_1, t_2)$ où t_1 et t_2 sont deux termes de domaines $D(t_1) = D(t_2) = \{a, b\}^\omega$, et $R_C = \{(\alpha_1, \alpha_2) \in D(t_1) \times D(t_2) \mid \forall n \in \mathbb{N}, |\alpha_1(0, n)|_a < |\alpha_2(0, n)|_a\}$, où $|w|_a$ représente le nombre de a présents dans $w \in \{a, b\}^*$. Cette contrainte impose que tout préfixe fini de α_1 contienne moins de a que le préfixe de α_2 de même longueur. L'affectation $\alpha = (bbb\ldots, aaa\ldots)$ satisfait cette contrainte, car le nombre de a dans n'importe quel préfixe de $bbb\ldots$ est toujours nul. Il est cependant difficile de représenter l'ensemble des affectations qui satisfont C autrement que par la définition de la contrainte elle-même. Il s'agit d'une contrainte globale au sens des flux, c'est-à-dire qui porte sur l'ensemble des datons de chaque terme.

Contraintes de voisinage

Nous définissons la notion de contrainte de voisinage. Une contrainte de voisinage s'applique localement sur les flux qui sont dans sa portée. Il s'agit toujours de l'application d'une relation n-aire sur un tuple de n termes, à la différence que la relation décrit un sous-ensemble du produit cartésien des alphabets des domaines des variables de sa portée, c'est-à-dire que pour une contrainte de voisinage $C : R_C(t_1, t_2, \ldots, t_n)$, $R_C \subseteq d(t_1) \times d(t_2) \times \cdots \times d(t_n)$.

Il est impossible de contraindre l'ensemble des datons des flux avec un nombre fini de contraintes de voisinage. C'est pourquoi nous définissons un mécanisme de réplication pour ce type de contrainte, qui permet de contraindre l'ensemble des datons d'une manière structurée.

Le plus simple est de considérer qu'une contrainte de voisinage est satisfaite si et seulement si elle est satisfaite en tout point du système de flux. Formellement, une contrainte de voisinage $C : R_C(t_1, t_2, \ldots, t_n)$ est satisfaite par une affectation $(\alpha_1, \alpha_2, \ldots, \alpha_n) \in D(t_1) \times D(t_2) \times \cdots \times D(t_n)$ si et seulement si :

$$\forall i \geq 0, (\alpha_1[i], \alpha_2[i], \ldots, \alpha_n[i]) \in R_C$$

Exemple Soient X et Y deux variables flux, dont les domaines sont $D(X) = D(Y) = \{a, b\}^\omega$, et $C_E : R_E(X, Y, \text{Next}(X))$ une contrainte de voisinage sur X et Y, qui impose sur Y la lettre la plus fréquente dans X et Next(X). La table de R_E est donnée dans la figure 1. Dans le cas où les deux lettres sont différentes, chacune peut être considérée comme la plus fréquente. Le réseau de contraintes correspondant à l'application de C_E sur X, Y et Next(X) est schématisé dans la figure 2. Dans cette contrainte, la valeur courante de Y dépend de la valeur future de X à tout instant.

Un exemple de solution est $\sigma = (\sigma(X), \sigma(Y))$, où $\sigma(X) = (aab)^\omega$ et $\sigma(Y) = (abbaba)^\omega$. En revanche, des flux dont les débuts sont respectivement $\sigma(X) = \langle a, a, b, b, a, b, \ldots \rangle$ et $\sigma(Y) = \langle b, b, b, b, b, b, b, \ldots \rangle$ ne peuvent pas satisfaire la contrainte, puisque le flux $\sigma(Y)$ doit commencer par un a étant donné $\sigma(X)$.

3.5 StCSPs

Un problème de Satisfaction de Contraintes sur Flux (StCSP) $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est la donnée d'un ensemble de variables flux $\mathcal{X} = (X_1, X_2, \ldots, X_n)$, d'un ensemble de domaines $\mathcal{D}$ et d'un ensemble de contraintes $\mathcal{C}$ sur ces variables. Le domaine d'une variable $X \in \mathcal{X}$ est noté $D(X)$. Il s'agit de l'ensemble des séquences infinies qui peuvent être construites sur l'alphabet $d(X)$ associé à la variable X, c'est-à-dire $D(X) = d(X)^\omega$.

R_E		
a	a	a
a	a	b
a	b	b
b	a	a
b	b	a
b	b	b

FIGURE 1 – Relation associée à l'exemple de contrainte de voisinage C_E

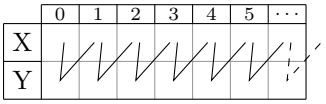

FIGURE 2 – Réseau de contraintes correspondant à l'application de C_E

$\mathcal{C}$ est un ensemble fini de contraintes de voisinage sur des termes. Les variables, les termes et les contraintes de voisinage ont été décrits dans les sections précédentes.

Une solution à un StCSP est une affectation $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \mathcal{D}(X_1) \times D(X_2) \times \cdots \times \mathcal{D}(X_n)$ qui satisfait toutes les contraintes.

La *temporalité* d'un StCSP est définie comme étant le plus grand nombre d'itérations de l'opérateur "Next" parmi tous les termes figurant dans le problème. Ainsi, un StCSP de temporalité 1 ne contient que des termes de la forme X ou Next(X), où X est une variable flux. Dans ce qui suit, un k-StCSP désigne un StCSP de temporalité k.

Une variable est un terme de temporalité 0. Un exemple de terme de temporalité 3 est $\text{Next}^3(X)$, où X est une variable flux. Enfin, $C : R_C(X, \text{Next}^2(X), \text{Next}(Y))$ est un exemple de contrainte de temporalité 2, où R_C décrit un sous-ensemble arbitraire de $d(X) \times d(X) \times d(Y)$.

4 Représentation de l'ensemble des solutions

Il est possible de représenter l'ensemble des solutions d'un StCSPs par une structure de taille finie, qui peut être calculée par un solveur de CSPs tel que Gecode. Nous présentons la méthode pour les StCSPs de temporalité 1. Nous présenterons dans la section suivante une méthode qui permet de transformer les StCSPs de temporalité plus grande en StCSP de temporalité 1.

Les contraintes d'un 1-StCSP peuvent être de trois formes :

Contraintes point-à-point Une contrainte point-à-point est une contrainte dans laquelle tous les termes sont de temporalité 0, c'est-à-dire qu'elle n'utilise pas l'opérateur "Next". Il s'agit d'une contrainte sur le produit cartésien des alphabets des variables apparaissant dans la contraintes. L'ensemble des contraintes de temporalité 0 représente un CSP classique dont toutes les solutions peuvent être calculées selon les méthodes habituelles. Ce type de contrainte permet de définir, par exemple, les états autorisés d'un système réactif.

Contraintes mixtes Une contrainte mixte est une contrainte dans laquelle certains termes sont de temporalité 0, et d'autres sont de temporalité 1. Il s'agit d'une contrainte de transition, car elle connecte la valeur courante des variables apparaissant dans les termes de temporalité 0, avec la valeur suivante des variables apparaissant dans les termes de temporalité 1. Ce type de contrainte permet de spécifier, par exemple, les règles d'évolution d'un automate.

Contraintes singulières Les contraintes singulières sont des contraintes dont aucun terme n'est de temporalité 0. Dans le cas des 1-StCSPs, si aucun terme n'est de temporalité 0, alors tous les termes sont de temporalité 1. Une telle contrainte correspond à une contrainte point-à-point qui s'applique partout sauf au premier point du système de flux, d'où le nom.

4.1 Système de transitions associé à un 1-StCSP

Un 1-StCSP qui ne contient que des contraintes de voisinage peut être transformé en un système de transitions.

Un système de transitions $\mathcal{T} = (Q, \delta)$ est un ensemble d'états Q accompagné d'une relation de transition $\delta \subseteq Q \times Q$. Un chemin dans $\mathcal{T}$ est une séquence $\langle q_1, q_2, \ldots, q_n \rangle$ de n états tels que $(q_i, q_{i+1}) \in \delta$ pour $0 < i \leq n$. La définition s'étend de façon naturelle aux chemins de longueurs infinies.

Pour présenter la transformation d'un StCSP en système de transitions, quelques définitions et notations sont nécessaires. Soit $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un StCSP où :

- $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ est un ensemble de n variables flux
- $\mathcal{D} = \{d(X_1)^\omega, d(X_2)^\omega, \ldots, d(X_n)^\omega\}$ est l'ensemble des domaines associés aux variables, où $d(X_i)$ est l'alphabet associé à la variable flux X_i
- $\mathcal{C} = \mathcal{C}^0 \cup \mathcal{C}^1$
- $\mathcal{C}^0 = \{C_1^0, C_2^0, \ldots, C_{k_0}^0\}$ est l'ensemble des k_0 contraintes point-à-point de $\mathcal{S}$
- $\mathcal{C}^1 = \{C_1^1, C_2^1, \ldots, C_{k_1}^1\}$ est l'ensemble des k_1 contraintes mixtes ou singulières de $\mathcal{S}$

Une solution de $\mathcal{S}$ est une affectation $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in d(X_1)^\omega \times d(X_2)^\omega \times \cdots \times d(X_n)^\omega$ qui satisfait toutes les contraintes, et peut être vue comme une séquence infinie de n-tuples $\tau \in d(X_1) \times d(X_2) \times \cdots \times d(X_n)$. Nous notons $\tau[\alpha]$ la séquence de tuples associée à α. Elle peut être définie récursivement en utilisant l'opérateur "Next" :

$$\tau[\alpha] = (\alpha_1(0), \alpha_2(0), \ldots, \alpha_n(0))\tau[\text{Next}(\alpha)]$$

Une solution de $\mathcal{S}$ correspond alors à la trace d'une exécution d'un système de transitions $\mathcal{T}(\mathcal{S}) = (Q, \delta)$, pour lequel $Q \subseteq d(X_1) \times d(X_2) \times \cdots \times d(X_n)$, et $(q_1, q_2) \in \delta$ si et seulement si il existe une solution α et un facteur $\alpha(i, i+1)$ de α tel que $q_1 = \tau[\alpha](i)$ et $q_2 = \tau[\alpha](i+1)$.

L'ensemble des états est l'ensemble des combinaisons qui satisfont toutes les contraintes point-à-point :

$$Q = \{\tau \in d(X_1) \times \cdots \times d(X_n) \mid \forall C \in \mathcal{C}^0, \tau \vDash C\}$$

où $\tau \vDash C$ signifie que l'affectation τ satisfait C.

La relation de transition est construite avec les contraintes mixtes :

$$\delta = \{(\tau_0, \tau_1) \in Q \times Q \mid \forall C \in \mathcal{C}^1, (\tau_0, \tau_1) \vDash C\}$$

où $(\tau_0, \tau_1) \vDash C$ signifie qu'un couple de tuples satisfait C, c'est-à-dire que les termes de temporalité 0 correspondent à des valeurs prises dans τ_0 et les termes de temporalité 1 correspondent à des valeurs prises dans τ_1.

Il s'ensuit que toutes les traces d'exécution du système de transitions $\mathcal{T}(\mathcal{S})$ associé à un StCSP $\mathcal{S}$ de temporalité 1 correspondent à des solutions de $\mathcal{S}$, car, par construction, les contraintes point-à-point et les contraintes mixtes de $\mathcal{S}$ sont satisfaites.

4.2 Calcul de l'ensemble des solutions

Nous montrons dans cette section que l'ensemble des solutions d'un 1-StCSP $\mathcal{S}$ ne comportant que des contraintes de voisinage, c'est à dire le système de transition $\mathcal{T}(\mathcal{S})$ associé à $\mathcal{S}$, peut être modélisé par un problème de satisfaction de contraintes, et donc calculé par un solveur de CSP.

Soit $\mathcal{S} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un 1-StCSP défini comme précédemment.

Le CSP $\mathcal{P}(\mathcal{S})$ associé à $\mathcal{S}$ comporte $2n$ variables X_1^0, $\ldots$, X_n^0, $X_1^1, \ldots, X_n^1$ où les domaines des variables sont $D(X_i^0) = D(X_i^1) = d_i$ et représentent respectivement la valeur de X_i à l'instant courant et la valeur de X_i à l'instant suivant.

Les contraintes point-à-point sont appliquées sur l'instant courant et l'instant suivant. Pour chaque contrainte $C_i(X_{j_1}, \ldots, X_{j_a}) \in \mathcal{C}^0$ d'arité a, les contraintes correspondantes dans $\mathcal{P}(\mathcal{S})$ sont $C_i(X^0_{j_1}, \ldots, X^0_{j_a})$ et $C_i(X^1_{j_1}, \ldots, X^1_{j_a})$.

Les contraintes mixtes sont appliquées partiellement à l'instant courant et partiellement à l'instant suivant. Les termes de temporalité 0 correspondent aux variables de l'instant courant, et les termes de temporalité 1 correspondent aux variables de l'instant suivant. Une contrainte mixte $C_i(\text{Next}^{b_1}(X_{j_1}), \ldots, \text{Next}^{b_a}(X_{j_a})) \in \mathcal{C}^1$ d'arité a dans $\mathcal{S}$ correspond à la contrainte $C_i(X^{b_1}_{j_1}, \ldots, X^{b_a}_{j_a})$ dans le CSP, où $b_k \in \{0, 1\}$ correspond à la temporalité du terme correspondant (1 si l'opérateur "Next" est utilisé et 0 sinon).

Les solutions de $\mathcal{P}(\mathcal{S})$ correspondent à l'ensemble des transitions de $\mathcal{T}(\mathcal{S})$ car les contraintes point-à-point sont satisfaites à chaque instant, et les contraintes mixtes sont satisfaites pour chaque transition. Il est donc possible de calculer le système de transitions $\mathcal{T}(\mathcal{S})$ associé à un 1-StCSP $\mathcal{S}$ en résolvant le problème de satisfaction de contraintes $\mathcal{P}(S)$ correspondant.

5 Réduction de temporalité

Cette section décrit comment un StCSP de temporalité supérieure à 1 peut être transformé en un 1-StCSP équivalent. Le principe consiste à ajouter de la mémoire au StCSP, en utilisant des variables auxiliaires qui contiennent les valeurs des variables aux instants futurs. Ces variables sont reliées entre elles par des contraintes d'égalité. De cette façon, à chaque instant, le système peut contenir de l'information sur les valeurs futures de certaines de ses variables.

Soient $\mathcal{S}$ un k-StCSP et $\mathcal{S}_R$ le 1-StCSP équivalent. $\mathcal{S}_R$ peut être calculé par itération de la procédure suivante :

Soit $\text{Next}^k(X)$ un terme de temporalité k. Soient X^{k-1} une nouvelle variable de domaine $D(X^{k-1}) = D(X)$, et $X^{k-1} = \text{Next}^{k-1}(X)$ une nouvelle contrainte d'égalité. Alors toutes les occurences du terme $\text{Next}^k(X)$ peuvent être remplacées par $\text{Next}(X^{k-1})$, qui est de temporalité 1. La contrainte d'égalité est de temporalité $k-1$. Si X était la seule variable de temporalité k dans le problème, alors la temporalité du problème a effectivement été réduite de 1. Sinon, la procédure doit être appliquée une nouvelle fois sur une variable différente. Comme le nombre de variables est fini, cette procédure finira par produire un StCSP de temporalité $k-1$, et finalement un StCSP de temporalité 1. La section précédente pourra alors être appliquée pour calculer l'ensemble des solutions.

Les StCSPs $\mathcal{S}$ et $\mathcal{S}_R$ sont équivalents au sens où les solutions de $\mathcal{S}_R$ sont identiques aux solutions de $\mathcal{S}$ lorsqu'elles sont projetées sur les variables de $\mathcal{S}$. En effet, les variables auxiliaires représentent des variables déjà présentes dans $\mathcal{S}$, mais à des instants différents. Dès lors qu'un solution est trouvée, les valeurs de ces variables deviennent redondantes et peuvent être éliminées de la représentation d'une solution. En revanche, ces variables sont nécessaires à la représentation de l'ensemble des solutions d'un k-StCSP.

5.1 Système de transitions étiqueté associé à un k-StCSP

Il est naturellement possible de représenter un k-StCSP $\mathcal{S}$ par le système de transitions associé au StCSP réduit $\mathcal{T}(\mathcal{S}_R)$. Cependant, ceci implique de représenter les variables auxiliaires de façon explicite, alors qu'elles ne sont qu'un artefact de la procédure utilisée pour obtenir une représentation finie de l'ensemble des solutions de $\mathcal{S}$. Les systèmes de transitions étiquetés permettent de représenter l'ensemble des solutions d'un k-StCSP d'une manière plus fidèle à sa spécification.

Une système de transitions étiqueté est un tuple $\mathcal{T}_E = (Q, \delta, E, L)$ où :
- Q est un ensemble d'états
- $\delta \subseteq Q \times Q$ est la relation de transition
- E est un ensemble d'étiquettes
- $L : Q \to E$ est une fonction d'étiquetage qui associe à chaque état une étiquette

Soit $\mathcal{T}_E(\mathcal{S})$ le système de transitions étiqueté associé à $\mathcal{S}$. Les états et la relation de $\mathcal{T}_E(\mathcal{S})$ sont exactement ceux de $\mathcal{T}(\mathcal{S}_R)$.

Les étiquettes correspondent à la projection de Q sur l'ensemble des variables de $\mathcal{S}$, et la fonction d'étiquetage effectue cette opération. Plus précisément, soit $q = (a_1, \ldots, a_n, a_{n+1}, \ldots, a_{n+m}) \in Q$, avec $(a_1, \ldots, a_n) \in d(X_1) \times \cdots \times d(X_n)$, où X_1, …, X_n sont les variables de $\mathcal{S}$, et $(a_{n+1}, \ldots, a_{n+m}) \in d(X_{n+1}) \times \cdots \times d(X_{n+m})$, où X_{n+1}, …, X_{n+m} sont des variables auxiliaires issues de la procédure de réduction de temporalité. Alors $L(q) = (a_1, \ldots, a_n)$.

6 Exemples

Nous présentons deux exemples de modélisation par des StCSPs. Le premier est un exemple très petit dont le thème est la synchronisation de feux de circulation

R_C	
V	O
V	R
O	V
R	V
O	O
R	R

R_{Ev}	
V	O
O	R
R	R
R	V

FIGURE 3 – Tables des relations utilisées par les contraintes de $\mathcal{S}_E$

sur un carrefour, qui permet d'expliciter les notions qui ont été introduites dans les sections précédentes. Le deuxième exemple est une adaptation du problème de configuration d'une chaîne de production automobile, qui est décrit dans CSPLib.

6.1 Carrefour

Cet exemple simplifié permet d'illustrer les applications des StCSPs pour la modélisation de problèmes temporels. Il s'agit de modéliser l'évolution de feux de circulation à un carrefour. Le problème $\mathcal{S}_E = \{\mathcal{X}_E, \mathcal{D}_E, \mathcal{C}_E\}$ comprend deux variables $\mathcal{X}_E = \{X, Y\}$ de domaines $D(X) = D(Y) = \{V, O, R\}$, qui représentent les états de deux feux de signalisation, où V symbolise un feu vert, O symbolise un feu orange et R symbolise un feu rouge. Les contraintes permettent de modéliser les états autorisés pour le système, ainsi que, de façon indépendante, les évolutions possibles pour chaque feu, dont la couleur évolue au cours du temps.

Nous définissons une relation R_C, dite "relation de compatibilité", pour spécifier les états autorisés du système, dont la table est donnée dans la figure 3. Cette relation permet de définir une contrainte point-à-point sur les feux. Dans le cas où le carrefour utiliserait plusieurs feux, il suffirait d'appliquer la relation sur chaque paire de feux pris deux à deux. Dans cet exemple, il n'y a que deux feux, donc une seule contrainte est nécessaire :

$$C_C : R_C(X, Y)$$

Comme il est pratique d'utiliser des tables pour représenter les relations, nous distinguons la relation de la contrainte elle-même, qui représente l'application de sa relation à un tuple de termes. Ici, la contrainte C_C représente l'applications de la relation R_C aux termes X et Y.

Les évolutions des feux sont spécifiées par la relation R_{Ev}, qui permet de définir des contraintes mixtes

X^0	Y^0	X^1	Y^1
V	R	O	R
O	R	R	V
O	R	R	R
R	V	R	O
R	O	V	R
R	O	R	R
R	R	V	R
R	R	R	V
R	R	R	R

FIGURE 4 – Solutions de $\mathcal{P}(\mathcal{S}_E)$

E_X et E_Y qui décrivent l'évolution des feux X et Y respectivement.

$$E_X : R_{Ev}(X, \text{Next}(X))$$

$$E_Y : R_{Ev}(Y, \text{Next}(Y))$$

CSP associé à $\mathcal{S}_E$ Le problème est de temporalité 1, il n'est donc pas nécessaire d'appliquer la procédure de réduction de temporalité, et le CSP associé à $\mathcal{S}_E$ est une traduction directe de $\mathcal{S}_E$ en CSP. Aucune variable n'est ajoutée, donc les variables du CSP sont $\mathcal{X} = \{X^0, Y^0, X^1, Y^1\}$. Les domaines sont $D(X^0) = D(X^1) = D(Y^0) = D(Y^1) = \{V, O, R\}$.

Les contraintes point-à-point sont appliquées sur chaque groupe, ce qui donne deux contraintes pour la contrainte de compatibilité C_C :

$$C_0 : R_C(X^0, Y^0)$$

$$C_1 : R_C(X^1, Y^1)$$

Les contraintes mixtes sont traduites directement.

$$E_X : R_{Ev}(X^0, X^1)$$

$$E_Y : R_{Ev}(Y^0, Y^1)$$

L'ensemble des solutions de ce CSP est donné par la figure 4, et correspondent au système de transitions de la figure 5.

Tous les chemins de $\mathcal{T}(\mathcal{S}_E)$ correspondent à une solution de $\mathcal{S}$. Un exemple de début de solution est :

X	V	O	R	R	V	O	R	...
Y	R	R	V	O	R	R	R	...

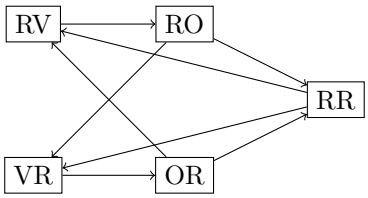

FIGURE 5 – Système de transitions $\mathcal{T}(\mathcal{S}_E)$ associé à $\mathcal{S}_E$

c_1	o_1	$\neg o_2$	o_3	o_4	$\neg o_5$
c_2	$\neg o_1$	$\neg o_2$	$\neg o_3$	o_4	$\neg o_5$
c_3	$\neg o_1$	o_2	$\neg o_3$	$\neg o_4$	o_5
c_4	$\neg o_1$	o_2	$\neg o_3$	o_4	$\neg o_5$
c_5	o_1	$\neg o_2$	o_3	$\neg o_4$	$\neg o_5$
c_6	o_1	o_2	$\neg o_3$	$\neg o_4$	$\neg o_5$

FIGURE 6 – Classes et options utilisées pour $\mathcal{S}_V$

6.2 Production automobile

Le problème de configuration d'une chaîne de production automobile (problème n°1 de CSPLib) est un problème classique de satisfaction de contraintes. Nous présentons une adaptation de ce problème au cadre des StCSPs.

Dans ce problème, une séquence infinie de voitures doit être produite, de telle sorte que les machines utilisées pour l'installation des options ne soient jamais surchargées. Les voitures peuvent être de différentes classes, en fonction des options qu'elles utilisent. Les options sont modélisées par des variables binaires, indépendantes les unes des autres.

Les options sont installées par des machines appropriées. Certaines options sont plus difficiles à mettre en place que d'autres, ce qui fait que pour un nombre donné de voitures, peu de voitures pourront recevoir une option difficile à installer. Cette notion de difficulté est encodé par une capacité k/n, qui exprime que pour n voitures consécutives, au plus k voitures pourront recevoir l'option.

Nous considérons une instanciation du problème avec 6 classes de voitures $\{c_1, c_2, \ldots, c_6\}$ qui sont des combinaisons de 5 options $\{o_1, o_2, \ldots, o_5\}$. La table est données sur la figure 6, ou o_k signifie que l'option o_k est installée, et $\neg o_k$ signifie que que l'option n'est pas installée pour la classe correspondante.

Les contraintes de capacités pour chaque option sont données dans la table 7.

Modélisation Nous notons $\mathcal{S}_V = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ la modélisation StCSP du problème présenté dans les paragraphes précédents. Nous utilisons cinq variables flux

o_1	o_2	o_3	o_4	o_5
1/2	2/3	1/3	2/5	1/5

FIGURE 7 – Contraintes de capacité pour les options de $\mathcal{S}_V$

O_1, O_2, O_3, O_4 et O_5, de domaines $D(O_1) = D(O_2) = D(O_3) = D(O_4) = D(O_5) = \{0,1\}^\omega$.

À tout instant, la combinaison des valeurs courantes des options doit correspondre à une des classes définies. Il s'agit d'une contrainte extensionnelle qui représente la figure 6.

$$
\begin{aligned}
(O_1, O_2, O_3, O_4, O_5) \in \{ \\
&(1,0,1,1,0), \\
&(0,0,0,1,0), \\
&(0,1,0,0,1), \\
&(0,1,0,1,0), \\
&(1,0,1,0,0), \\
&(1,1,0,0,0) \\
\}
\end{aligned}
$$

En ce qui concerne les contraintes de capacité, il est possible d'exploiter les contraintes habituellement utilisées en programmation par contraintes. Les cinq contraintes suivantes permettent d'exprimer les containtes de capacité.

$$
\begin{aligned}
&O_1 + \text{Next}(O_1) \leq 1 \\
&O_2 + \text{Next}(O_2) + \text{Next}^2(O_2) \leq 2 \\
&O_3 + \text{Next}(O_3) + \text{Next}^2(O_3) \leq 1 \\
&O_4 + \text{Next}(O_4) + \text{Next}^2(O_4) + \\
&\qquad \text{Next}^3(O_4) + \text{Next}^4(O_4) \leq 2 \\
&O_5 + \text{Next}(O_5) + \text{Next}^2(O_5) + \\
&\qquad \text{Next}^3(O_5) + \text{Next}^4(O_5) \leq 1
\end{aligned}
$$

Ce problème est de temporalité 4. Après réduction de temporalité, le problème contient 8 variables auxiliaires, ce qui génère un CSP de 26 variables. Le systèmes de transition correspondant est suffisamment grand pour ne pas être inclus dans la page. Cependant, les systèmes générés par ce problème et ses variantes ne sont en général pas très grands comparés à ce qu'il est possible de manipuler avec des systèmes informatisés. Ceci rend ce problème particulièrement approprié à l'étude des StCSPs.

7 Recherche incrémentale

Il n'est pas nécessaire de calculer le système de transitions entièrement pour trouver une solution. La modélisation sous forme de CSP peut être adaptée de telle

Algorithme : Recherche incrémentale

$H \leftarrow \emptyset$
Pour chaque solution (τ_1, τ_2) de $\mathcal{P}$
  ajouter τ_1 et τ_2 à H
  $\tau \leftarrow$résoudre($\mathcal{P}$,τ_2)
  Si $\tau \neq \bot$
    afficher τ_2
    afficher τ_1
    afficher τ
    sortir
  Sinon
    enlever τ_1 et τ_2 de H
  Fin Si
Fin Pour

Procédure résoudre($\mathcal{P}, \tau_1$)
  Pour chaque solution τ_2 de $\mathcal{P}[\tau_1]$
    Si $\tau_2 \in H$
      renvoyer τ_2
    Sinon
      ajouter τ_2 à H
      $\tau \leftarrow$résoudre($\mathcal{P}$,τ_2)
      Si $\tau \neq \bot$
        afficher τ_2
        renvoyer τ
      Fin Si
      supprimer τ_2 de H
    Fin Si
  Fin Pour
  renvoyer $\bot$
Fin Procédure

FIGURE 8 – Résolution incrémentale d'un StCSP

sorte qu'une partie du système de transitions seulement soit calculée.

Un solveur de CSPs capable de générer l'ensemble des solutions d'un problème doit être capable de mémoriser la position courante dans l'espace de recherche de manière à pouvoir reprendre la recherche là où elle s'est arrêtée une fois que la solution courante a été affichée. Il est naturellement possible de faire plus que cela, notamment de résoudre un autre CSP paramétré par la solution trouvée.

L'algorithme présenté dans la figure 8 utilise une liste de hachage H pour mémoriser l'ensemble des états trouvés, afin de pouvoir détecter un cycle. La variable τ est utilisée pour repérer le cycle, qui est un état qui doit être déjà présent dans H. La valeur spéciale $\bot$ signifie que τ n'est pas assigné. Si τ est différent de $\bot$, cela signifie qu'une séquence $\tau_1\tau_2 \ldots \tau_k$ satisfaisant toutes les contraintes a été trouvée, avec $\tau = \tau_i$ pour $0 \leq i \leq k$, ce qui représente un cycle dans le système de transitions.

L'algorithme travaille sur le CSP $\mathcal{P}(\mathcal{S}_R)$ associé au StCSP réduit, noté $\mathcal{P}$. Ce problème est composé d'un groupe de variables pour les valeurs à l'instant courant, et d'un groupe de variables pour les valeurs à l'instant suivant. L'algorithme commence par itérer l'ensemble des transitions. Pour chaque solution trouvée, il appelle la procédure `résoudre`. La notation $\mathcal{P}[\tau]$ désigne le problème $\mathcal{P}$ dans lequel le groupe de variables correspondant à l'instant courant est fixé à τ.

La solution est contenue dans la pile d'appel au moment où `résoudre` renvoit autre chose que $\bot$. L'algorithme produit un affichage de la solution, par l'intermédiaire des instructions `afficher`, qui doit être lu de bas en haut. L'état nécessaire pour déterminer le cycle est affiché en dernier.

Comme le nombre de solutions d'un CSP est fini, chaque boucle termine. La profondeur d'appel est également bornée par le nombre d'états du système de transitions. Ceci implique que l'algorithme finit toujours par s'arrêter. La complexité de cet algorithme est optimale en espace, car l'espace requis est linéaire en la taille de la solution trouvée. En revanche, il ne garantit pas de trouver une solution qui soit la plus courte. Il n'est pas non plus optimal en temps, car certains chemins du systèmes de transitions peuvent être explorés plusieurs fois, mais ceci peut être corrigé en utilisant une structure de données appropriée pour mémoriser les états qui mènent à des impasses. Enfin, une absence de cycle dans le systèmes de transitions implique que l'ensemble du système aura été parcouru. Le pire des cas est un système qui contient un maximum de transitions et aucun cycle, ce qui n'est pas fréquent, car dans la plupart des cas, un système de transitions contient soit peu de transitions, soit beaucoup de cycles.

8 Conclusion et perspectives

Nous avons présenté dans cet article un nouveau cadre pour la programmation par contraintes sur des variables flux. La figure 9 schématise les relations entres les différentes parties.

De nombreux formalismes existent concernant le raisonnement temporel. Les logiques temporelles sont très utilisées en vérification des modèles [4], mais les approches à basées contraintes sont virtuellement inexistantes. En particulier, bien que la transformation d'un k-StCSP en formule de logique temporel linéraire soit possible, elle rend impossible l'exploitation de la sémantique des contraintes.

Une littérature assez diverse existe en dehors du champ de la vérification de modèles. En particulier,

un formalisme a été développé concernant la satisfaction de formules de logique temporelle linéaire avec contraintes sur des structures particulières [3]. Bien que l'approche initiale dans [3] soit semblable, les structures, les types de contraintes et la manière de les résoudres sont très différents. En particulier, l'article s'appuie sur les méthodes habituellement utilisées pour la satisfaction de formules de logiques temporelles, qui ne sont pas basées sur la programmation par contraintes.

Les travaux de programmation par contraintes qui utilisent des contraintes temporelles sont très nombreux, mais sont dans la majorité des cas basés sur une approche avec horizon qui doit être repoussé indéfiniment, donnant lieu à des instances de plus en plus grandes du même problème[1].

Pour autant que nous puissions en juger, ce travail fait partie des premiers travaux qui concernent l'application de la programmation par contraintes à la modélisation de problèmes temporels sans horizon. Ce travail est fait en collaboration avec l'Université Chinoise de Hong Kong, et le formalisme décrit dans cet article est par conséquent fondamentalement similaire à celui décrit dans [7].

Le cadre des StCSPs étant nouveau, beaucoup de problèmes restent ouverts. Les possibilités concernant la modélisation restent à explorer, et les possibilités d'applications sont encore à l'étude, mais semblent nombreuses. Des sources d'inspiration sont les domaines de la planification, du contrôle, et de l'interaction, qui ne sont pas des domaines habituellement rencontrés en programmation par contraintes.

L'efficacité des algorithmes dans la pratique reste encore à mesurer. Des directions de recherches concernent le calcul d'une représentation compactes de l'ensemble des solutions basées sur les BDDs [8], mais le problème de l'explosion du nombre d'états est probablement beaucoup moins apparent en ce qui concerne la recherche d'une solution particulière.

Enfin, l'ensemble de toutes les solutions n'est pas toujours nécessaire. Le calcul d'un petit ensemble de solutions permettant de générer des solutions non périodiques serait suffisant pour des applications musicales, où la monotonie d'une solution unique, nécessairement périodique, serait d'un faible intérêt.

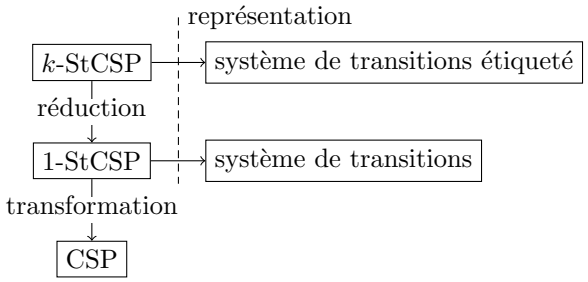

FIGURE 9 – Relations entre les formalismes

Références

[1] Krzysztof R. Apt and Sebastian Brand. Constraint-based qualitative simulation. *CoRR*, abs/cs/0504024, 2005.

[2] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems.* International Series in Operations Research and Management Science. Kluwer, 2001.

[3] Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint ltl. *Inf. Comput.*, 205(3) :380–415, 2007.

[4] E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)*, pages 995–1072. 1990.

[5] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice.* Elsevier, 2004.

[6] Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux. Constraint programming for mining n-ary patterns. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 552–567. Springer, 2010.

[7] Arnaud Lallouet, Yat-Chiu Law, Jimmy Ho-Man Lee, and Charles F. K. Siu. Constraint programming on infinite data streams. In Toby Walsh, editor, *IJCAI*, pages 597–604. IJCAI/AAAI, 2011.

[8] K. L. Mcmillan. *Symbolic Model Checking : An Approach to the State Explosion Problem.* PhD thesis, Carnigie Mellon University, 1992.

[9] Cédric Pralet and Gérard Verfaillie. Using constraint networks on timelines to model and solve planning and scheduling problems. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric A. Hansen, editors, *ICAPS*, pages 272–279. AAAI, 2008.