Etude de modèles de programmation par contraintes pour le problème du voyageur de commerce avec fenêtres de temps

Sylvain Ducomman[1 2] **Hadrien Cambazard**[1] **Bernard Penz**[1]

[1] Univ. Grenoble Alpes, G-SCOP, F-38000 Grenoble, France
CNRS, G-SCOP, F-38000 Grenoble, France
[2] Geoconcept SA, 92220 Bagneux, France
{prénom.nom}@grenoble-inp.fr sylvain.ducomman@geoconcept.com

Résumé

Le problème du voyageur de commerce avec fenêtres de temps (TSPTW) consiste à visiter un ensemble de clients, en respectant pour chaque client une fenêtre de temps donnée, tout en minimisant la distance totale parcourue. Ce problème est une extension du problème du voyageur de commerce qui est NP-difficile au sens fort. Il a été largement traité, entre autres par la Programmation Par Contraintes (CP). Dans cet article, nous présentons différents modèles par Programmation Par Contraintes. Nous proposons ensuite différentes approches dans le but d'améliorer la propagation, notamment en utilisant la contrainte globale WeightedCircuit. Enfin, nous étudions plusieurs stratégies de branchement ainsi qu'une technique simple d'enregistrement de nogood. L'évaluation expérimentale est réalisée sur des benchmarks de la littérature. Ce travail est une première étape pour évaluer des techniques de filtrage basées sur les coûts et traiter des problèmes de tournées de véhicules avec fenêtres de temps.

Abstract

The Traveling Salesman Problem with time-windows (TSPTW) is to visit a set customers within given time-windows while minimizing the overall distance of the tour. It is a widely studied extension of the Traveling Salesman Problem and has received some attention of the Constraint Programming (CP) community in the past. We investigate in this paper several CP models for the problem. In particular, we evaluate the interest of the WeightedCircuit global constraint. We also discuss branching strategies and a simple nogood recording technique. Results are presented on a well known benchmark of the TSPTW and aims at understanding the current performances of CP technology on this problem. This is a first step in a study of cost-based filtering techniques for routing problems with time-windows.

1 Introduction

Le Problème du Voyageur de Commerce (TSP) est un problème combinatoire largement étudié. Le problème rencontré par ce voyageur est de visiter chacun de ses clients une et une seule fois avant de revenir chez lui en parcourant une distance minimale. Dans cette étude, nous nous intéressons à une extension du TSP prenant en compte les fenêtres de temps et le temps de service des clients. Il s'agit du Problème du Voyageur du Commerce avec Fenêtres de Temps (TSPTW). L'objet du TSPTW est de trouver le plus court chemin visitant les clients une et une seule fois tout en respectant leurs fenêtres de temps. Cela signifie que les clients doivent être visités dans l'intervalle correspondant. Il est possible d'arriver chez un client avant sa date de disponibilité et d'attendre pour effectuer le service au début de sa fenêtre de temps. Le TSPTW est un sous-problème important au sein d'une large classe de problèmes dits de tournées de véhicules. Il correspond également à un problème d'ordonnancement sur une seule machine avec un temps de set-up dépendant de la séquence et des dates de disponibilités et de fins des tâches. Le TSPTW est évidemment au moins aussi difficile que le TSP et c'est donc un problème NP-Difficile [21]. Dans cette étude, nous étudions le TSPTW avec une approche Programmation Par Contraintes (CP). Nous évaluons différents modèles réalisant une propagation plus ou moins forte. Nous nous intéresserons, par ailleurs, à différentes stratégies de recherche et examinons le meilleur compro-

mis entre les modèles et les stratégies de recherche. Cette étude est utile pour identifier le couple modèle/stratégie le plus robuste afin de l'inclure dans un solveur de problèmes de tournées de véhicules.
Après une brève introduction sur les travaux déjà effectués (section 2), nous proposons d'abord trois modèles en CP (section 4). Nous examinons ensuite des techniques de propagation afin d'améliorer le filtrage (section 5). Enfin, différentes stratégies de recherche sont abordées (section 6). Dans la dernière section (section 7), les résultats seront présentés et comparés.

2 État de l'art

L'une des premières approches exactes pour résoudre le TSPTW est proposée par [8]. Elle est basée sur une procédure de Branch & Bound dont la borne inférieure est obtenue à l'aide de la programmation dynamique. Une approche par Branch & Bound est également présentée dans [2] avec une borne inférieure s'appuyant sur le dual de la formulation proposée. Ces approches minimisent le temps total et non le temps de trajet qui a été introduit dans [16] avec un programme linéaire où l'on retrouve une formulation à base de flots. Des contraintes de précédences sont aussi utilisées et couplées avec une méthode de réduction des fenêtres de temps proposée dans [10]. Cette méthode réduit les bornes des fenêtres de temps en examinant les successeurs et prédécesseurs ainsi que la contrainte de temps. Dans [1] et plus récemment dans [9], les auteurs résolvent la version asymétrique du TSPTW par une méthode de Branch-and-Cut. Plusieurs techniques de propagation sont aussi proposées, des pré-traitements sur les données ou encore des heuristiques pour obtenir une première solution. À ce jour l'une des meilleures approches exactes s'appuie sur une programmation dynamique utilisant des bornes inférieures obtenues par génération de colonnes [3].
En Programmation Par Contraintes, deux approches ont été étudiées. La première est celle de [20] dans laquelle les auteurs proposent des contraintes redondantes permettant de réduire l'espace de recherche. En particulier, ils utilisent la réduction des fenêtres de temps couplée avec une contrainte d'élimination des arcs. Cette dernière met à jour systématiquement deux ensembles pour chaque noeud représentant les noeuds pouvant se placer avant ou après le noeud considéré. Les auteurs utilisent aussi une stratégie de recherche efficace qui est basée sur la fréquence d'apparition d'un client dans le domaine des successeurs ou prédécesseurs. Dans la seconde approche [12], les auteurs se basent sur le modèle de [20] et ajoutent des bornes inférieures utilisant un problème d'affectation. Ils utilisent cette borne inférieure pour réaliser un filtrage par les coûts et l'améliorent avec une méthode de plans coupants.
Des heuristiques\metaheuristiques ont aussi été proposées afin d'obtenir des résultats rapidement sans avoir de preuve d'optimalité. Dans [13], une heuristique d'insertion est proposée. On peut également citer [6] qui utilisent un algorithme de recuit simulé, version améliorée dans [19]. Plus récemment, un algorithme Beam-ACO [5, 18] est développé pour le TSPTW. De plus, les auteurs proposent une formalisation des instances de TSPTW dans un format standard.

3 Problème et notations

On considère un graphe orienté complet $G(N, E)$ dans lequel E est l'ensemble des arcs et $N = \{0, \ldots, n+1\}$ est l'ensemble des noeuds représentant les clients et le dépôt. On distingue de plus les ensembles suivants $N^s = \{0, \ldots, n\}$ et $N^e = \{1, \ldots, n+1\}$). Les noeuds 0 et $n+1$ correspondent au dépôt : le noeud 0 est le dépôt d'origine où la route commence, le noeud $n+1$ est le dépôt de destination où la route se termine. Nous associons une fenêtre de temps $[a_i, b_i]$ pour chaque noeud $i \in N^e$. On considère que le voyageur part à l'instant $t = 0$ du noeud 0 et que la fenêtre du noeud $n+1$ est donc du type $[0, b_{n+1}]$ indiquant que le chemin doit être complété avant l'heure b_{n+1}. Une distance $d_{ij} \in \mathbb{N}$ et un temps $t_{ij} \in \mathbb{N}$ de trajet sont associés à chaque arc (i, j). Le temps de service du client i est inclus, sans perte de généralité, dans chaque temps de trajet t_{ij} vers les noeuds $j \in N - \{i\}$.

4 Modèles

Par la suite $\overline{x}$ (resp. $\underline{x}$) désigne la borne supérieure (resp. la borne inférieure) de la variable x et $D(x)$ indique son domaine (ensemble fini de valeurs possibles pour x).

4.1 Premier modèle

Le modèle de base en Programmation Par Contraintes s'appuie sur les variables $next_i$ pour chaque noeud i. $next_i \in [1, \ldots, n+1]$ représente le successeur immédiat du noeud i dans la tournée (par convention $next_{n+1} = 0$). De façon symétrique, la variable $pred_i \in [0, \ldots, n]$ représente le prédécesseur du client i dans la tournée ($pred_0 = n+1$). Des variables représentant une accumulation de quantité complètent le modèle : $dist$ et $start$. $dist_i \in [0, \ldots, M]$, représente pour chaque noeud i la distance parcourue en arrivant au noeud i tandis que $start_i \in [a_i, \ldots, b_i]$, représente le temps (ou l'heure) d'arrivée chez le client i. M représente ici une borne supérieure de la distance totale

de parcours.

Le modèle de base (M_1) est le suivant :

$$\textbf{Minimize } z \tag{1}$$

$$z = \sum_{i=0}^{n}(d_{i,next_i}) = \sum_{i=1}^{n+1}(d_{pred_i,i}) \tag{2}$$

$$\textsc{Circuit}(next_0, ..., next_{n+1}) \tag{3}$$

$$(M_1) \quad dist_{next_i} = dist_i + d_{i,next_i} \qquad \forall i \in N^s \tag{4}$$

$$dist_0 = 0 \tag{5}$$

$$start_{next_i} \geq start_i + t_{i,next_i} \qquad \forall i \in N^s \tag{6}$$

$$start_0 = 0 \tag{7}$$

La contrainte (3), présentée dans [17], permet de maintenir un circuit dans le graphe en visitant une seule fois chaque noeud. Cette contrainte implique notamment $\textsc{AllDifferent}(next_0, \ldots, next_{n+1})$. Les contraintes (4) et (6) s'appuient sur l'utilisation de la contrainte Element [24] afin de maintenir l'accumulation de quantités le long de la route. Les variables *pred* sont liées aux variables *next* par la propriété suivante : $next_i = j \Leftrightarrow pred_j = i$ pour chaque paire (i, j). Cette propriété se traduit en terme de Programmation Par Contraintes par l'utilisation de la contrainte globale Inverse :

$$\textsc{Inverse}([next_0, \ldots, next_{n+1}], [pred_0, \ldots, pred_{n+1}]) \tag{8}$$

L'utilisation des prédecesseurs renforce la borne inférieure sur z car les quantités $\sum_{i=0}^{n}(\underline{d_{i,next_i}})$ et $\sum_{i=1}^{n+1}(\underline{d_{pred_i,i}})$ ne sont pas forcément égales ce qui explique les contraintes d'égalité (2). Cependant ces quantités sont égales lorsque les variables sont instanciées. De plus ces variables offrent des possibilités supplémentaires pour le branchement.

4.2 Contraintes redondantes

En CP, l'utilisation de contraintes redondantes permet le renforcement de la propagation. La contrainte sur l'élimination des arcs proposée dans [16], permet l'élimination de valeurs dans le domaine des *next* en considérant les fenêtres de temps.

$$start_i + t_{ij} > start_j \Rightarrow next_i \neq j \quad \forall (i, j) \in N^s \times N^e \; i \neq j \tag{9}$$

Cette dernière contrainte est couplée avec la contrainte de réduction des fenêtres de temps présentée dans [10]. En effet, les domaines des successeurs et prédécesseurs peuvent être utilisés pour réduire les fenêtres de temps.

$$start_i \geq \min_{k \in D(pred_i)}(start_k + t_{ki}) \qquad \forall i \in N^e \tag{10}$$

$$start_i \leq \max_{k \in D(next_i)}(start_k - t_{ik}) \qquad \forall i \in N^s \tag{11}$$

On notera que la contrainte (6) n'est pas équivalente en termes de propagation. Les dernières contraintes présentées sont utilisées dans [20]. Ces raisonnements peuvent être renforcés en remplaçant les temps de trajets t_{ab} par le temps du plus court chemin entre a et b et en considérant non plus les successeurs et prédécesseurs directs mais les clients devant être visités après ou avant. Toutefois, dans [12], les expérimentations ont montré que le calcul du plus court chemin était trop lourd pour le gain obtenu.

Nous présentons maintenant deux modèles permettant d'améliorer le modèle de base.

4.3 Modèle booléen

On ajoute à (M_1) des variables booléennes b_{ij} pour chaque couple (i, j). Ces dernières permettent d'identifier l'ordre relatif d'un client i par rapport à un client j : $b_{ij} = 1$ si le client i est visité avant le client j et 0 sinon. Ainsi au lieu de raisonner uniquement sur les successeurs/prédécesseurs directs, on s'intéresse aux clients situés après/avant dans le chemin. Les contraintes associées à ce modèle sont les suivantes $\forall (i, k, j) \in N^3$ s.t $i \neq k \neq j$:

$$b_{ik} + b_{ki} = 1 \tag{12}$$

$$(b_{ik} = 1 \wedge b_{kj} = 1) \Rightarrow b_{ij} = 1 \tag{13}$$

$$(M_2) \quad (b_{ik} = 1) \Rightarrow start_k \geq start_i + t_{ik} \tag{14}$$

$$(b_{ik} = 1) \wedge (b_{kj} = 1) \Rightarrow next_i \neq j \tag{15}$$

$$(b_{ik} = 1) \Rightarrow next_k \neq i \tag{16}$$

Nous soulignons ici la contrainte (15) dont l'apport peut être important (voir aussi [20]). L'inconvénient de ce modèle est que le nombre de contraintes est en $\mathcal{O}(n^3)$. Ces raisonnements peuvent constituer l'objet d'une contrainte globale manipulant l'ensemble du graphe de précédences pour éviter un problème de mémoire. Cette contrainte peut en particulier assurer la fermeture transitive faite par (13) et la propagation vers les variables *next* faite par (15). Nous examinons un modèle intermédiaire dans la section suivante.

4.4 Modèle position

Nous conservons l'esprit général du modèle booléen, mises à part les contraintes (13) et (15) qui sont trop nombreuses. Les variables b_{ij} sont toujours présentes mais nous ajoutons des variables pos_i représentant la position du noeud i dans la tournée :

$$pos_i = \sum_{j \in N \setminus \{i\}} b_{ji} \qquad \forall i \in N \tag{17}$$

Les contraintes de ce modèle sont les suivantes, $\forall (i,k) \in N^2$ s.t $i \neq k$:

$$b_{ik} + b_{ki} = 1 \quad (18)$$
$$(b_{ik} = 1) \Rightarrow start_k \geq start_i + t_{ik} \quad (19)$$
$$(M_3) \quad (b_{ik} = 1) \Rightarrow next_k \neq i \quad (20)$$
$$pos_k > pos_i + 1 \Rightarrow next_i \neq k \quad (21)$$
$$pos_k > pos_i \Leftrightarrow b_{ik} = 1 \quad (22)$$
$$\text{AllDifferent}(pos_0, \ldots, pos_{n+1}) \quad (23)$$

Ainsi la contrainte (15) est remplacée par (22). On notera que le filtrage obtenue par (22) est plus faible que celui de (15).

Ce modèle est adapté à la mise en oeuvre de raisonnements énergétiques semblables à ceux de la contrainte Disjunctive. Notre expérience est que l'application du filtrage de la Disjunctive est trop coûteux dans ce contexte et pas toujours utile. Nous proposons ici un raisonnement qui nous semble un bon compromis entre temps et filtrage. Il s'agit de mettre à jour une variable de position pos_i par rapport à l'ensemble des autres clients et de leurs possibilités de placement avant ou après le client i.

Soit A_i l'ensemble des visites qui peuvent être placées après le client i et B_i l'ensemble des clients pouvant être avant ou après le client i :

$$A_i = \{j \in N \mid \underline{b_{ij}} = 1\}$$
$$B_i = \{j \in N \mid \overline{b_{ij}} = 1 \wedge \underline{b_{ij}} = 0\}$$

Le temps minimum requis après i est noté EA_i :

$$EA_i = \underline{t_{i,next_i}} + \sum_{j \in A_i} \underline{t_{j,next_j}}$$

Dans un premier temps, on note que la borne supérieure de $start_i$ ne peut pas dépasser le temps minimum requis par les visites de A_i :

$$\overline{start_i} \leq \overline{start_{n+1}} - EA_i$$

Dans un second temps, nous filtrons la borne inférieure de pos_i. Considérons les visites $x_1, \ldots, x_k$ de B_i rangées par ordre croissant de distances aux successeurs :

$$\underline{t_{x_1,next_{x_1}}} \leq \underline{t_{x_2,next_{x_2}}} \leq \underline{t_{x_3,next_{x_3}}} \leq \ldots \leq \underline{t_{x_k,next_{x_k}}}$$

Soit c le plus grand entier tel que :

$$EA_i + \sum_{j=0}^{c} \underline{t_{x_j,next_{x_j}}} \leq (\overline{start_{n+1}} - \underline{start_i})$$

$c + |A_i|$ est donc le nombre de visites maximum qui peuvent avoir lieu après la visite i, ce qui nous donne une borne inférieure pour pos_i :

$$\underline{pos_i} \geq n - (c + |A_i| + 1)$$

Un raisonnement symétrique est effectué pour la borne inférieure de $start_i$ et la borne supérieure de pos_i en prenant en compte le nombre minimum de visites pouvant se placer avant le client i.

Ce raisonnement est un cas particulier de raisonnement énergétique que ferait la contrainte Disjunctive [7] par l'algorithme d'*edge-finding*. Comme mentionné en introduction, le TSPTW peut se voir comme un problème d'ordonnancement sur une machine avec un temps de setup dépendant de la séquence. On peut écrire un modèle CP s'appuyant sur une Disjunctive avec des durées variables pour les tâches. En ajoutant une variable end_i telle que $start_i + t_{i,next_i} = end_i$. Ce modèle n'est pas présenté plus en détails car sa mise en oeuvre n'a pas été concluante pour le moment.

En ignorant le raisonnement énergétique proposé sur les positions, on peut ranger les modèles selon le niveau de propagation atteint. En effet, le modèle de base ($M1$) possède un niveau de propagation plus faible comparé au modèle booléen ($M2$) qui possède la plus forte propagation. Le modèle position ($M3$) est un modèle intermédiaire en terme de propagation.

5 Amélioration de la propagation

Les modèles précédents souffrent de l'absence d'une borne inférieure globale de la fonction objectif. Nous proposons donc de remplacer la contrainte Circuit par la contrainte WeightedCircuit présentée dans [4]. Cependant, les techniques coûteuses de filtrage orientées par le coût se rentabilisent d'autant plus que la borne supérieure (en cas de minimisation) est une borne de qualité au noeud racine. En effet, tous les raisonnements s'appuient sur l'écart à la meilleure solution connue au moment de leur application. Nous proposons donc de calculer une borne supérieure de l'objectif avant la recherche par une heuristique simple.

5.1 Borne supérieure

Pour obtenir une borne supérieure au noeud racine, nous effectuons une recherche locale très simple. Deux opérateurs de voisinage sont appliqués sur une séquence (initialement aléatoire) de clients : l'échange de deux clients ou le déplacement d'un client dans la séquence. Le premier mouvement améliorant trouvé est effectué. Un mouvement est améliorant s'il réduit le nombre de clients hors de leur fenêtre de temps ou s'il réduit la distance parcourue sans augmenter le nombre de clients visités à l'extérieur de leur fenêtre (optimisation lexicographique). L'algorithme termine en atteignant un minimum local, c'est à dire une séquence

de clients non améliorable par ces deux opérateurs. K exécutions de cet algorithme sont effectuées à partir de séquences aléatoires et la meilleure solution réalisable est utilisée pour initialiser la borne supérieure de l'objectif.

5.2 Weighted Circuit

La seconde technique employée est le remplacement de la contrainte CIRCUIT par la contrainte WEIGHTEDCIRCUIT de [4]. Cette contrainte propage une borne inférieure de z et filtre des valeurs des variables *next* du point de vue du coût. La WEIGHTEDCIRCUIT permet de maintenir un circuit dans un graphe pondéré tout en considérant la minimisation du poids total. Elle correspond au problème du Voyageur du Commerce :

$$\text{WEIGHTEDCIRCUIT}([next_0, \ldots, next_{n+1}], z)$$

L'algorithme de filtrage est basé sur la relaxation du TSP qui utilise la "1-tree relaxation" introduite par Held et Karp [14]. La "1-tree relaxation" relâche la contrainte de degré du TSP et utilise la structure du 1-tree. Cette dernière correspond à un arbre dans le graphe contenant les noeuds $\{1, \ldots, n+1\}$ et à deux arcs incidents au noeud 0. Nous cherchons, dans cette relaxation, un 1-tree de poids minimum. Un tour dans le TSP est un cas particulier de 1-tree. La borne de Held et Karp s'obtient par relaxation Lagrangienne [14, 15] (pour une étude de la borne de Held et Karp, se référer à [23]).
La relaxation lagrangienne est basée sur le potentiel de chaque noeud et le coût lié au potentiel pour chaque arc. Le potentiel d'un noeud représente le coût de violation de la contrainte de degré. Ainsi, plus le potentiel d'un noeud est élevé, plus le coût d'un arc incident sera élevé. Par conséquent, à chaque itération l'algorithme aura tendance à converger vers un 1-tree de poids minimum respectant la contrainte de degré. Lors de l'algorithme de filtrage, nous pouvons distinguer deux phases : l'identification des arcs interdits et l'identification des arcs obligatoires.

Identification des arcs interdits Le calcul du coût marginal d'un arc e est nécessaire pour l'identification des arcs interdits. Ce coût représente l'augmentation du coût du 1-tree si e devait être un arc du 1-tree de poids minimum. Ainsi, si le coût global du 1-tree de poids minimum plus le coût marginal d'un arc e est plus grand que la borne supérieure de z, alors e est un arc interdit et donc il faut le supprimer du graphe.

Identification des arcs obligatoires. L'identification des arcs obligatoires est similaire à l'identification des arcs interdits. Le coût de remplacement d'un arc e représente le coût supplémentaire du 1-tree de poids minimum lorsque l'arc e est remplacé par un arc ne faisant pas partie du 1-tree. Si le coût de remplacement d'un arc e associé au coût du 1-tree est supérieur à la borne supérieure de z, alors l'arc e est un arc obligatoire. Cependant, nous ne pouvons pas encore utiliser ce résultat car notre problème de base est un problème dans un graphe orienté. Nous devons pour cela, vérifier le domaine des variables *next* associé à l'arc e afin d'identifier si une arête est obligatoire.

6 Stratégies de recherche

Dans cette section, nous abordons différentes stratégies de recherche pour les modèles présentés précédemment. Nous examinons des schémas de branchement consistant à affecter une variable à une valeur de son domaine.

Plusieurs stratégies applicables à tous les modèles sont présentées ainsi que deux stratégies spécifiques aux modèles booléen (M_2) et position (M_3).

6.1 Stratégies de recherche génériques

PathMaintain : Cette stratégie consiste à étendre un chemin partant du noeud 0. Après avoir affecté $next_i$ à j, on sélectionne donc la variable $next_j$ pour prolonger le chemin. La valeur choisie est le noeud ayant sa fenêtre de temps la plus proche.

MinDomNextPred : Ce schéma de branchement est souvent utilisé en CP. La variable de plus petit domaine parmi toutes les variables *next* et *pred* est choisie en priorité. La valeur choisie est la plus petite valeur dans le domaine de la variable sélectionnée.

HeuristiquePesant : L'heuristique de Pesant est le schéma de branchement utilisé dans [20].

1. Soit s la taille du plus petit domaine des *next* et *pred*.
   Soit $\mathcal{V} = \{next_i | \ |D(next_i)| = s, \forall i = 1, \ldots, n\} \cup \{pred_i | \ |D(pred_i)| = s, \forall i = 1, \ldots, n\}$.
2. Si $|\mathcal{V}| = 1$, choisir la variable contenue dans $\mathcal{V}$
3. Sinon
   (a) Pour chaque élément e dans $\cup_{v \in \mathcal{V}} D(v)$, calculer $e^{\#}$ le nombre d'apparition de e dans le domaine des variables de $\mathcal{V}$.
   (b) Choisir la variable qui maximise $f(v) = \sum_{e \in D(v)} e^{\#}$.

Nous affectons par la suite la valeur correspondant au client le plus proche (en distance).

NoGoodsRecording : Un nogood est *une affectation partielle qui ne peut pas être étendue à une solution faisable*. Ainsi, toute affectation contenant un nogood est soit irréalisable du point de vue des fenêtres de temps, soit de coût supérieur à $\overline{z}$. L'enregistrement de nogoods peut permettre d'éviter l'exploration redondante de sous-arbres de recherche. L'approche proposée ici impose un schéma de branchement s'appuyant sur PathMaintain. Une affectation des variables *next* depuis le noeud dépôt constitue donc un chemin partiel. Cette affectation est définie par l'ensemble $S \subset N$ de clients visités sur ce chemin, le **dernier** client noté k ($k \in S$), la distance d parcourue jusqu'au dernier client et le temps t indiquant le service au plus tôt du client k. On caractérise donc une affectation partielle de l'heuristique PathMaintain par le quadruplet (S, k, d, t). Si l'affectation (S_1, k_1, d_1, t_1) est un nogood, alors il est inutile d'essayer d'étendre toute affectation (S_2, k_2, d_2, t_2) telle que :

$$S_2 = S_1,\ \ k_2 = k_1,\ \ d_2 \geq d_1 \ \ \text{et} \ \ t_2 \geq t_1$$

Sachant que (S_1, k_1, d_1, t_1) est un nogood, il est inutile d'explorer un chemin partiel contenant les mêmes clients S (pas forcément dans le même ordre), terminant au même noeud k_1 et arrivant plus tard en k_1 tout en ayant parcouru une distance plus longue.
On enregistre les nogoods au backtrack pendant le déroulement de l'heuristique PathMaintain. De plus, à partir du dernier client i sur le chemin partiel, le client $j \in D(next_i)$ est éliminé du domaine de $next_i$ si le chemin partiel obtenu est prouvé irréalisable par un nogood connu.
L'heuristique de branchement PathMaintain permet de se placer dans le même espace de recherche que l'approche Programmation Dynamique proposée dans [11]. L'enregistrement des nogoods permet de couper certaines branches de l'espace de recherche cité précédemment.

6.2 Stratégie de recherche pour le modèle booléen

Dans cette section, nous présentons une stratégie de recherche dédiée au modèle booléen (section 4.3).

ImpactBoolVar : Cette stratégie de recherche consiste à mesurer l'impact de branchement des b_{ij} sur la réduction des fenêtres de temps. Cet impact correspond à la somme des réductions potentielles sur les domaines des variables *start* lorsque la variable est égale à 0 ou 1. La variable provoquant la plus forte réduction est choisie en priorité.

1. Pour chaque paire de noeud (i, j), les réductions de domaine sont notées $startGain(i, j)$, $endGain(i, j)$, $startGain(j, i)$, $endGain(j, i)$ selon que $b_{ij} = 1$ ou $b_{ij} = 0$
   — $startGain(i, j) = \max(0, \underline{start_i} + t_{ij} - \underline{start_j})$
   — $endGain(i, j) = \max(0, \overline{start_i} - (\overline{start_j} - t_{ij}))$
2. Sélectionner la variable b_{ij} qui maximise $f(i, j) = startGain(i, j) + endGain(i, j) + startGain(j, i) + endGain(j, i)$
3. Appliquer $b_{ij} = 1$

6.3 Stratégie de recherche pour le modèle position

Cette section présente une stratégie de recherche dédiée au modèle position (section 4.4).

InflectionPoint : Le principe de cette heuristique est d'appliquer l'heuristique MinDomNextPred sur des groupes de variables identifiés au noeud racine après la propagation initiale. Ces groupes constituent des ensembles de positions indépendants. En effet, on remarque qu'une position instanciée au noeud racine sépare le problème en deux ensembles de positions disjoints.

7 Résultats

Les résultats sont séparés en deux parties. La première partie est l'analyse des techniques de renforcement de la propagation (section 5). La seconde partie est l'analyse des couples modèle/stratégie de recherche. Cette analyse est une première base pour le développement d'une relaxation prenant en compte les fenêtres de temps ainsi que les distances (coûts). Le développement est fait en c++ avec la librairie or-tools [25] pour la Programmation Par Contraintes. Les tests ont été effectués sur un Intel Xeon 4 coeurs 2.27 GHz avec 8.00 Go de mémoire avec une limite de temps de 1200 secondes.

7.1 Amélioration de la propagation

Le but de cette section est de montrer l'apport des différentes techniques utilisées afin d'améliorer la propagation (section 5). Les instances utilisées sont les neufs premières instances de Pesant *et al.* [20]. Ce sont des instances dérivées des problèmes RC2 de Solomon [22] pour le problème VRPTW. Le modèle utilisé est le modèle booléen et la stratégie de recherche est MinDomNextPred. Toutes les instances sont résolues à l'optimum. Deux critères d'évaluation sont utilisés, le temps de résolution en seconde et le nombre d'échecs dans l'arbre de recherche.

La Figure 1 présente les résultats obtenus sur les neufs instances pour les différentes techniques. Basis représente le modèle de base sans ajouts. UpperBound est l'ajout de la borne supérieure sur le modèle de base. Nous pouvons voir que cet ajout apporte déjà beaucoup pour filtrage. WeightedCircuit est le remplacement de Circuit par WeightedCircuit. Même

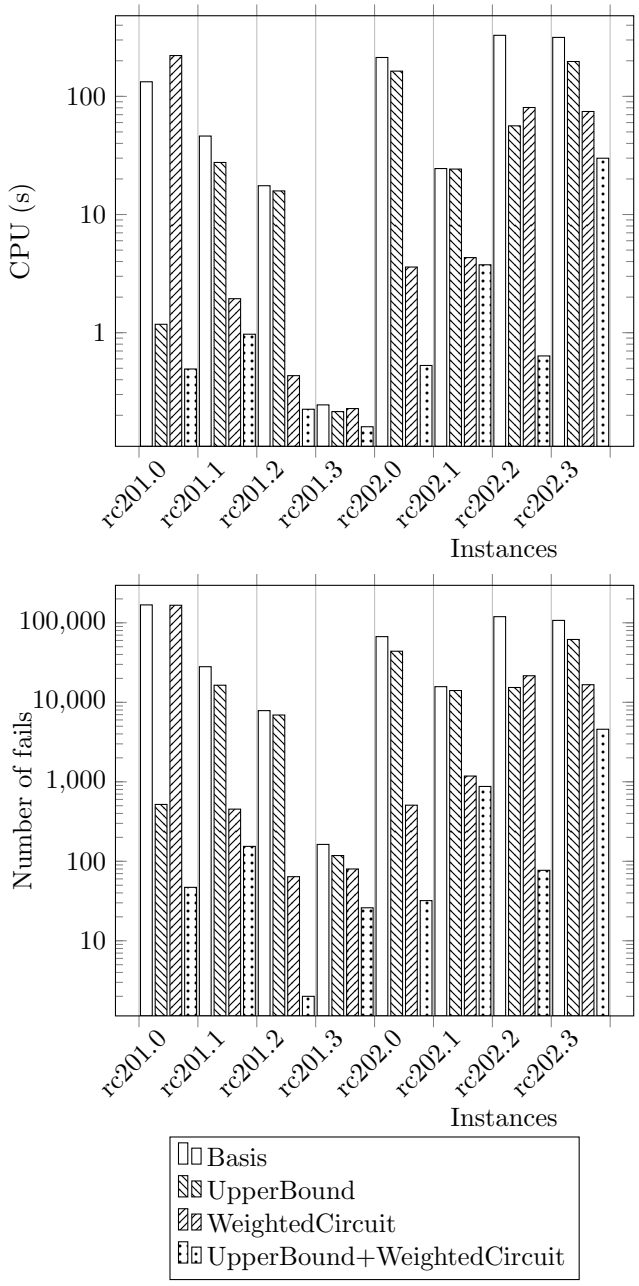

FIGURE 1 – Amélioration de la propagation

si l'algorithme de filtrage est lourd (voir rc201.0), le gain reste significatif. Cependant, l'ajout des deux techniques UpperBound+WeightedCircuit est encore plus performant et permet de réduire significativement l'arbre de recherche.

7.2 Résultats des différents modèles et stratégies de recherche

Dans cette section, nous présentons les différentes performances des couples modèle/stratégie de recherche. Dans un premier temps, nous verrons le gain apporté par les NogoodsRecording sur la stratégie de recherche PathMaintain. Dans un second temps, l'ensemble des couples sera étudié. Au vu des résultats de la section précédente, nous décidons de rajouter les techniques de propagation à tous les modèles (base, booléen, position). Les instances utilisées sont les instances 20 et 40 clients de Dumas [11]. Chaque instance est divisée en groupe de 5 problèmes (w20, w40, w60, w80, w100). La différence entre les groupes est la largeur des fenêtres de temps. Pour les problèmes w20, les fenêtres de temps sont très petites entrainant une propagation importante sur l'ordre des visites. En revanche, avec les problèmes w100, les fenêtres de temps sont beaucoup plus larges.

PathMaintain/NoGoods : La Figure 2 présente les différents modèles (base, booléen et position) couplés aux stratégies de recherche PathMaintain et NogoodsRecording sur les instances à 20 clients.

Nous nous apercevons que le modèle de base n'est pas compétitif par rapport aux deux autres modèles. Au delà de w60, le modèle de base n'arrive pas à résoudre les problèmes. Évidement, le modèle position filtre beaucoup moins que le modèle booléen (le nombre d'échecs pour le modèle position est supérieure au modèle booléen). Cependant, pour des problèmes avec des fenêtres de temps petites (w20, w40) le modèle position est plus rapide que le modèle booléen. De plus, le schéma de branchement NogoodsRecording entraîne une diminution du nombre d'échecs par rapport à la stratégie de recherche PathMaintain. Cependant, cette dernière n'est pas aussi performante que d'autres schémas de branchement.

Couple modèle/stratégie de recherche : La Figure 3 montre les différents couples modèle/stratégie de recherche sur les instances 40 clients pour des petites fenêtres de temps (w20, w40) et les instances 20 clients pour le reste.

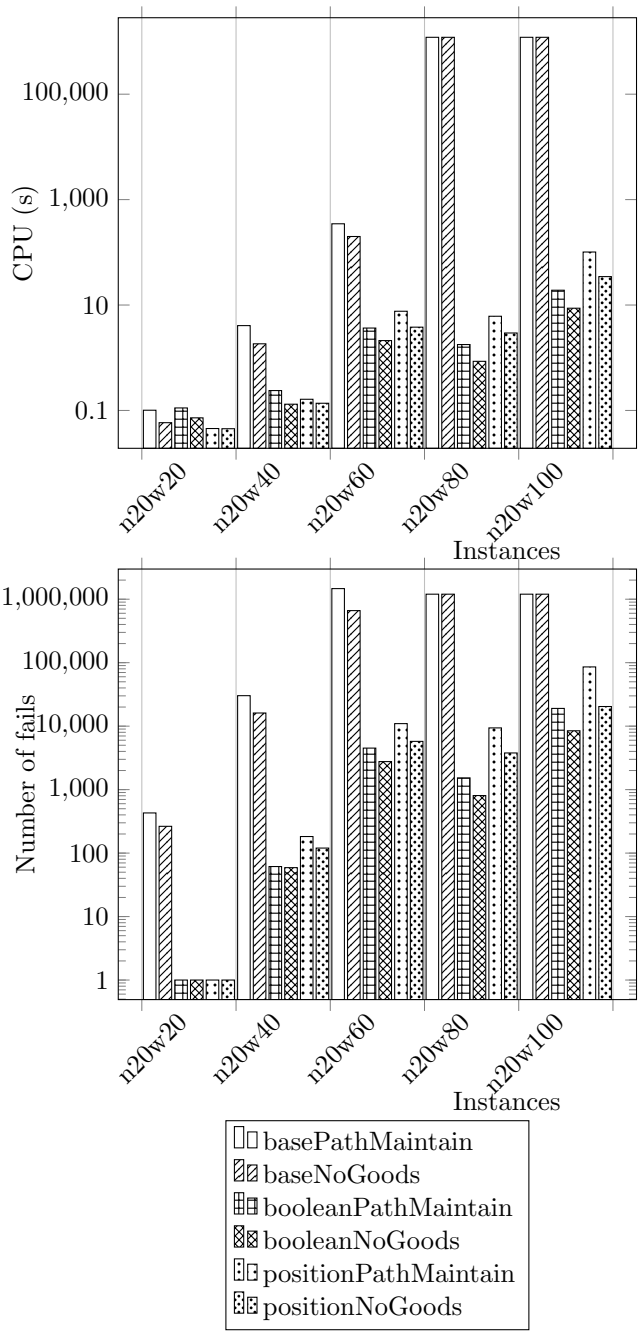

FIGURE 2 – Différence entre modèles/PathMaintain ou NoGoodsRecording

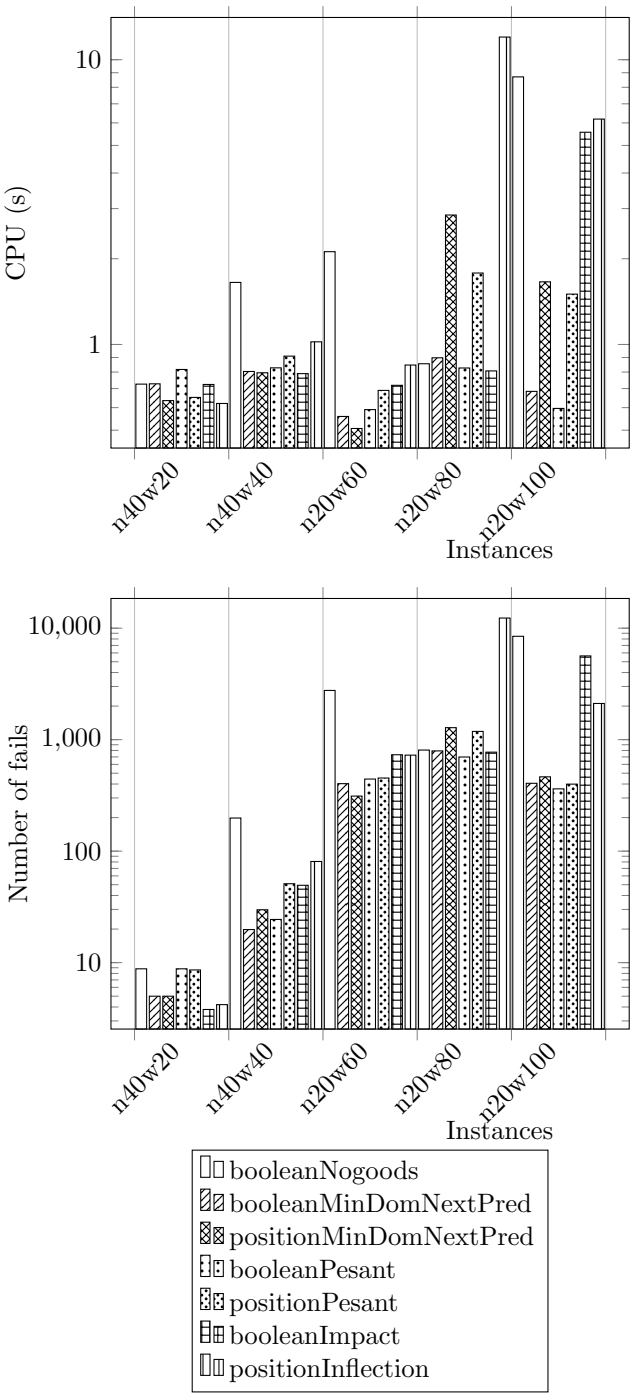

FIGURE 3 – Différence entre modèles/stratégies de recherche

Instances	Résultats	position/HeuriticPesant		
	[20], [12]	Solution	Temps(s)	# d'échec
rc201.0	628.62^{p}	628.62	0.59	97
rc201.1	654.70^{p}	654.7	0.70	150
rc201.2	707.65^{p}	707.65	0.17	2
rc201.3	422.54^{p}	422.54	0.10	23
rc202.0	496.22^{p}	496.22	0.56	50
rc202.1	426.53^{p}	426.53	2.95	950
rc202.2	611.77^{p}	611.77	5.60	958
rc202.3	627.85^{p}	627.85	38.72	7573
rc203.0	.	.	1200	198002
rc203.1	.	.	1200	104930
rc203.2	.	617.46	148.8	24266
rc204.0	.	541.45	1.47	79
rc204.1	.	485.37	0.49	38
rc204.2	.	.	1200	74351
rc205.0	511.65^{p}	511.65	4.90	1452
rc205.1	491.22^{p}	491.22	0.16	14
rc205.2	714.69^{f}	715.34*	1200	418318
rc205.3	601.24^{p}	601.24	38.14	10909
rc206.0	.	835.23*	1200	129235
rc206.1	.	664.73	2.06	254
rc206.2	655.37^{p}	655.37	6.42	1219
rc207.0	806.69^{f}	806.69	953.93	37390
rc207.1	726.36^{f}	726.36	8.60	521
rc207.2	546.41^{p}	546.41	0.90	55
rc208.0	.	.	1200	48194
rc208.1	.	509.04	7.77	914
rc208.2	.	503.92	0.38	4

^{p} : Optimal value from [20], ^{f} : Optimal value from [12],* : Optimality not proved

Tableau 1 – Résultats des instances proposées par Pesant *et al.* [20] avec le modèle position/heuristique de Pesant

Tout d'abord, nous nous apercevons que les stratégies de recherche dédiées à un modèle ne sont pas performantes par rapport à l'heuristique Pesant *et al.* ou MinDomNextPred. Ensuite, le modèle position est efficace, en terme de temps de résolution, pour les problèmes avec des petites fenêtres de temps (w20, w40, w60). Cependant, lorsque les fenêtres de temps deviennent larges, la vision globale apportée par le modèle booléen (modèle avec le plus fort raisonnement de propagation) est plus performante. Enfin, la stratégie de recherche des NogoodsRecording ne semble pas prometteuse.

Comparaison avec les résultats de la littérature : Le Tableau 1 propose une comparaison des résultats avec les différentes études en Programmation Par Contraintes dans [20] et [12] sur les instances de Solomon et Pesant *et al.*.

Nous résolvons à l'optimum sept instances de plus que les approches proposées dans [20] et [12]. Toutefois, l'optimum de l'instance rc205.2 n'est pas atteint alors que [12] prouve l'optimalité. Les instances non résolues sont les instances avec un nombre de clients supérieur à 35. Notons que la totalité de ces benchmarks a été résolue par [3].

8 Conclusion

Nous avons tout d'abord étudié trois modèles en Programmation Par Contraintes pour le TSPTW possédant des niveaux de propagation différents. Nous avons ensuite constaté que ces modèles étaient fortement renforcés en remplaçant la contrainte Circuit par WeightedCircuit couplée au calcul d'une borne supérieure. Enfin, différentes stratégies de recherche ont été testées. Cependant, on s'aperçoit que les stratégies de recherche efficaces sont celles déjà largement utilisées dans la littérature (HeuristiquePesant, MinDomNextPred). Cette étude constitue un travail préliminaire pour évaluer l'intérêt d'un filtrage global orienté par les coûts pour le TSPTW. Notre objectif est de mettre en oeuvre ce filtrage sur des problèmes plus généraux de tournées de véhicules.

Références

[1] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by Branch-and-Cut. *Mathematical Programming*, 90(3) :475–506, 2001.

[2] Edward K. Baker. Technical note—an exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31(5) :938–945, 1983.

[3] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(3) :356–371, 2012.

[4] Pascal Benchimol, Willem-Jan Van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3) :205–233, 2012.

[5] Christian Blum. Beam-ACO for simple assembly line balancing. *INFORMS Journal on Computing*, 20(4) :618–627, 2008.

[6] Stephen P. Brooks and Byron J.T. Morgan. Optimization using simulated annealing. *The Statistician*, pages 241–257, 1995.

[7] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1) :42 – 47, 1982.

[8] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2) :145–164, 1981.

[9] Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1) :132–147, 2012.

[10] Martin Desrochers, Jacques Desrosiers, and Marius M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2) :342–354, 1992.

[11] Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2) :367–371, 1995.

[12] Filippo Focacci, Andrea Lodi, and Michela Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14(4) :403–417, 2002.

[13] Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3) :330–335, 1998.

[14] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6) :1138–1162, 1970.

[15] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees : Part II. *Mathematical Programming*, 1(1) :6–25, 1971.

[16] André Langevin, Martin Desrochers, Jacques Desrosiers, Sylvie Gélinas, and François Soumis. A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks*, 23(7) :631–640, 1993.

[17] Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1) :29–127, 1978.

[18] Manuel López-Ibáñez and Christian Blum. Beam-ACO for the travelling salesman problem with time windows. *Computers & Operations Research*, 37(9) :1570–1583, 2010.

[19] Jeffrey W. Ohlmann and Barrett W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1) :80–90, 2007.

[20] Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1) :12–29, 1998.

[21] Martin W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1) :285–305, 1985.

[22] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2) :254–265, 1987.

[23] Christine L. Valenzuela and Antonia J. Jones. Estimating the Held-Karp lower bound for the geometric TSP. *European Journal of Operational Research*, 102(1) :157–175, 1997.

[24] Pascal Van Hentenryck and Jean-Philippe Carillon. Generality versus specificity : An experience with AI and OR techniques. In *AAAI*, pages 660–664, 1988.

[25] Nikolaj van Omme, Laurent Perron, and Vincent Furnon. or-tools user's manual. Technical report, Google, 2014.