Une contrainte globale de bin-packing avec précédences : application au problème d'équilibrage de lignes d'assemblage

Pierre Schauss, Yves Deville

Une contrainte globale de bin-packing avec précédences: Application au problème d'équilibrage de lignes d'assemblage.

Pierre Schaus **Yves Deville**

Département d'ingénierie informatique,
Université de Louvain,
Place Sainte Barbe 2,
B-1348 Louvain-la-Neuve, Belgique

{pierre.schaus,yves.deville}@uclouvain.be

Résumé

Les problèmes d'équilibrage de lignes d'assemblage (PELA) sont d'une importance primordiale pour l'industrie depuis l'invention de la première ligne d'assemblage de la Ford T imaginée par Henry Ford. L'objectif haut niveau de ces problèmes est d'optimiser la conception des lignes tout en respectant diverses contraintes. Les contraintes de précédences entre les tâches sont omniprésentes dans les PELA. Plus concrètement, l'objectif est alors de répartir les tâches entre divers postes de travail de sorte que le taux de production soit maximal. Ce problème peut être modélisé comme un problème de bin-packing avec des contraintes de précédence (BPCP) où les bins sont les postes de travail et les objets à placer correspondent aux tâches. Paul Shaw a introduit une contrainte globale pour le bin-packing. Malheureusement cette contrainte ne prend pas en compte les précédences du BPCP. Dans cet article, nous introduisons dans un premier temps des contraintes redondantes pour le BBCP combinant les précédences et le bin-packing. Ces contraintes redondantes permettant de résoudre des instances non solubles sans celles-ci en programmation par contraintes. Nous introduisons également une contrainte globale pour le BPPC qui permet d'élaguer d'avantage l'arbre de recherche. Enfin, nous expérimentons notre modèle de PC pour le BBCP afin résoudre le PELA. Nous proposons également deux heuristiques et montrons l'efficacité de notre approche sur des jeux de données standards du PELA. En comparaisons aux techniques dédiées n'utilisant pas la PC, notre approche est plus flexible. En effet elle permet aisément l'ajout de nouvelles contraintes pouvant survenir dans des applications réelles.

1 Introduction

Une variante du problème de bin-packing est de placer des objets de différentes tailles dans un nombre fini de bins ayant une capacité fixe de sorte que la charge soit bien répartie entre les bins (e.g. minimiser la hauteur maximum des bins). Nous nous intéressons ici aux problèmes de bin-packing avec des contraintes de précédence entre les objets (BPCP). Les bins sont ordonnées et une contrainte de précédence entre l'objet o_1 et o_2 est satisfaite si l'objet o_1 est placé dans une bin B_1, et l'objet o_2 dans une bin B_2 avec $B_1 \leq B_2$.

Les problèmes de BPCP sont fréquents lors de la conception de ligne d'assemblage dans l'industrie [1]. Étant donné un ensemble de tâches de différentes durées, soumises à des contraintes de précédences et une constante appelée temps de cycle, le problème est de répartir les tâches entre les stations de travail le long de la ligne de production de sorte qu'aucune station n'obtienne un temps supérieur au temps de cycle pour accomplir toutes les tâches qui lui sont af-

[1] Voir par exemple les deux logiciels commerciaux Proplanner® *www.proplanner.com* et OptiLine® *www.optimaldesign.com*

fectées (temps de station), et de sorte que les contraintes de précédence soient satisfaites. Le problème de décision de répartir de manière optimale (équilibrage) les tâches dans les stations de travail par rapport à un objectif est appelé le problème d'équilibrage de ligne d'assemblage (PELA) [2, 14].

En particulier, lorsque le nombre de stations est fixé, le problème est de répartir les tâches dans les stations de sorte que le temps de station soit équilibré c'est à dire qu'il minimise le temps de cycle. Ce type de problème est communément appelé PELA simple de type 2 (PELAS-2) [2]. Il est évident que le PELAS-2 et le BPCP sont équivalents.

Le BPCP et le PELAS-2 peuvent être résolus par méthodes exactes ou heuristiques. Dans cet article nous nous focalisons sur les méthodes exactes. Les méthodes exactes sont généralement des algorithmes de branch-and-bound dédiés tels que Salome 2 [8]. Ces algorithmes sont extrêmement efficaces et sont régulièrement améliorés depuis environ 50 ans. Malheureusement, ces algorithmes dédiés sont très peu flexibles à l'ajout de nouvelles contraintes pouvant apparaître dans des applications réelles telles qu'une distance minimale entre deux tâches ou encore une restriction sur la valeur cumulée d'un attribut particulier des tâches (voir le classifieur de problèmes disponible sur *www.assembly-line-balancing.de*) pour plus de détails. Lorsque des contraintes de la sorte sont ajoutées, nous obtenons le dénommé problème généralisé d'équilibrage de ligne d'assemblage (PGELA) [1]. Les méthodes exactes ne sont pas suffisamment flexibles pour gérer le PGELA efficacement. Le paradigme de la programmation par contraintes (PC) est un bon candidat pour solutionner de tels problèmes. En effet, en PC des contraintes peuvent être ajoutées aisément au modèle.

La PC a déjà été utilisée avec succès pour le problème de bin-packing (voir [15]). Le problème d'équilibrage des horaires de cours est équivalent au BPCP. L'objectif est d'affecter des cours à un certain nombre de périodes tout en respectant les contraintes de pré-requis entre les cours et de sorte que le temps de travail soit équitablement répartit entre les périodes. Des modèles de PC basiques ont été proposés dans [3, 6].

Dans cet article, nous proposons un modèle PC pour le BPCP et le PELAS-2, permettant d'exprimer de nouvelles contraintes telles que dans le PGELA. Plus précisément, nos contributions sont :

- Un modèle de PC simple pour le bin-packing avec contraintes de précédences (BPCP).
- De nouvelles contraintes redondantes pour le BPPC basées sur des variables ensemblistes et la fermeture transitive du graphe de précédences.
- Une contrainte globale permettant de faire davantage de filtrage que les contraintes redondantes. L'algorithme de filtrage présenté est en $O(n^2)$ où n est le nombre d'objets (tâches).
- Une validation expérimentale sur des données standards pour le PELAS-2 montrant que les contraintes redondantes et la contrainte globale permettent de résoudre des instances qui ne sont pas solubles sinon. Nous montrons également la flexibilité de l'approche en ajoutant diverses contraintes pour former un PGELA.

La structure de l'article est la suivante. Une introduction à la programmation par contraintes est suivie d'un modèle de PC pour le BPCP. De nouvelles contraintes redondantes sont introduites. Ensuite une contrainte globale et son algorithme de filtrage est décrit. Finalement, avant de conclure, il y a une section expérimentale évaluant les performances de l'approche sur des jeux de données standards pour le PELA.

2 Introduction à la PC

La programmation par contraintes est un paradigme puissant pour résoudre des problèmes combinatoires de recherche (PCR). Un PCR est composé d'un ensemble de variables ; chaque variable ayant sont domaine de valeurs possibles, et d'un ensemble de contraintes sur les variables. L'objectif est de trouver une affectation des variables qui satisfait toutes les contraintes. Une fonction à optimiser peut également être ajoutée. La PC alterne une composante de recherche (backtracking ou branch-and-bound) avec une composante d'inférence (également appelée propagation ou filtrage) dont le but est de réduire l'espace de recherche en supprimant les valeurs ne participant dans aucune solution.

Étant donné une variable à domaine fini entière X ayant pour domaine $Dom(X)$, nous dénotons par $X^{\min}$ et $X^{\max}$ les valeurs minimum et maximum de son domaine. Les variables ensemblistes en PC [5] permettent de représenter des ensembles plutôt que des valeurs. Le domaine d'une variable ensembliste $\mathcal{S}$ est représenté par deux ensembles $\underline{\mathcal{S}}$ et $\overline{\mathcal{S}}$ avec $\underline{\mathcal{S}} \subseteq \overline{\mathcal{S}}$. La borne inférieure $\underline{\mathcal{S}}$ représente les valeurs qui doivent nécessairement faire partie de l'ensemble final alors que $\overline{\mathcal{S}}$ représente les valeurs qui feront potentiellement partie de l'ensemble. La relation $\underline{\mathcal{S}} \subseteq \mathcal{S} \subseteq \overline{\mathcal{S}}$ doit évidemment être satisfaite.

Les contraintes redondantes sont des contraintes déduites des contraintes définissant le problème. Elles ne sont pas à proprement parler nécessaires puisqu'elles n'altèrent pas l'ensemble des solutions. Elles sont donc logiquement redondantes. Néanmoins, ajouter de telles contraintes au modèle peut réduire fortement l'espace de recherche car un filtrage additionnel est possible. Les contraintes redondantes ont été appliquées avec succès à bon nombre de problèmes tels que le séquencement de voitures.

Une contrainte globale peut-être vue comme une contrainte sur un ensemble de variables, modélisant une une partie bien définie du problème et possédant son propre

algorithme de filtrage dédié [17].

3 Un modèle de PC pour le bin-packing avec contraintes de précédences.

Le bin-packing avec contraintes de précédences (BBCP) implique les paramètres et variables suivants :

- n valeur positives $[s_1, ..., s_n]$ représentant le poids de chaque objet.
- le nombre de bins disponibles m.
- m variables $[L_1, ..., L_m]$ représentant le poids de chaque bin ($Dom(L_i) = \{0, \ldots, \sum_i s_i\}$)
- n variables $[B_1, ..., B_n]$ représentant pour chaque objet la bin dans laquelle l'objet est placé ($Dom(B_i) = \{1, \ldots, m\}$)
- un graphe de précédences (dirigé et acyclique) $G(\{1, .., n\}, E)$

Les contraintes doivent modéliser les précédences ainsi que le poids des bins. L'objectif est de minimiser le poids maximum, *i.e.* $\text{minimiser} \quad \max\{L_i\}$.

Les contraintes de précédence se modélisent aisément avec un ensemble de contraintes :

$$B_i \leq B_j \quad (\text{pour } (i,j) \in E). \tag{1}$$

La manière la plus simple de modéliser le poids de chaque bin est d'introduire des variables binaires $X_{ij} \in \{0, 1\}$ disant si l'objet j est placé dans la bin i. Ces variables sont liées aux variables du problème à l'aide de contraintes réifiées :

$$X_{ij} = 1 \leftrightarrow B_j = i. \tag{2}$$

Le poids L_i de chaque bin est alors simplement le produit scalaire :

$$L_i = \sum_j X_{ij} \cdot s_j. \tag{3}$$

Nous améliorons ce modèle avec la contrainte redondante suivante [15] :

$$\sum_i L_i = \sum_j s_j. \tag{4}$$

La contrainte globale de bin-packing décrite dans [15] et dénommée `IloPack` dans Ilog Solver [7], peut également être ajoutée. Cette contrainte comprend également le filtrage effectué par la contrainte redondante (3).

Le gain de performance obtenu avec la contrainte redondante (4) et `IloPack` est illustré sur une petite instance du PELAS-2 du jeu de données de Scholl [13]. Le problème est de placer les objets (tâches) dans 12 bins (stations de travail) tout en satisfaisant les contraintes de précédence données à la Figure 1, et en minimisant le poids maximum des bins.

Le Tableau 1 montre (colonnes A,B and C) qu'ajouter la contrainte redondante (4) réduit significativement le nombre de backtracks et le temps, et que la contrainte de bin-packing de [15] améliore encore d'avantage les performances.

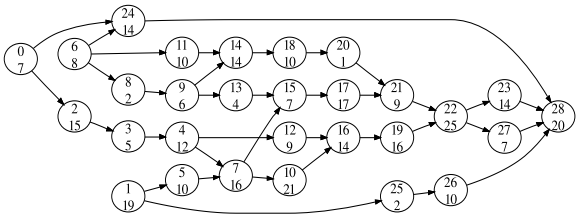

FIG. 1 – Graphe de précédences de l'instance Buxey du jeu de données de Scholl [13]. Pour chaque tâche, le nombre au dessus est son identifiant (attribué dans l'ordre topologique du graphe de précédence) et le nombre en dessous est sa durée.

A		B		C		D	
#bks	time	#bks	time	#bks	time	#bks	time
6894	10.62	4850	7.34	2694	4.3	445	1.07

TAB. 1 – Comparaison des modèles de bin-packing en termes de backtracks et temps (seconde) sur l'instance Buxey avec 12 stations de travail. La colonne A est obtenue avec le modèle basique, la colonne B en ajoutant la contrainte redondante (4), la colonne C en ajoutant la contrainte `IloPack` décrite dans [15] et la colonne D en ajoutant également les contraintes redondantes (5) et (7).

4 De nouvelles contraintes redondantes pour le BPCP

Nous introduisons de nouvelles contraintes redondantes améliorant le filtrage sur BPPC en liant les précédences avec le problème de bin-packing à l'aide de variables ensemblistes.

Nous notons $\mathcal{P}_i$ la variable ensembliste représentant les prédécesseurs de l'objet i dans le BPCP. Un objet j est prédécesseur d'un objet i si et seulement si l'objet j est positionné dans une bin précédente ou égale à la bin de l'objet i :

$$j \in \mathcal{P}_i \leftrightarrow B_j \leq B_i. \tag{5}$$

La borne inférieure $\underline{\mathcal{P}_i}$ est initialisée à i plus l'ensemble des objets ayant un arc pointant vers i dans la fermeture transitive du graphe de précédences. La fermeture transitive est calculée avec l'algorithme de Floyd Warshall en $O(n^3)$ (voir [4]). Le temps de pré-traitement nécessaire au calcul de la fermeture transitive pour l'instanciation des bornes inférieures $\underline{\mathcal{P}_i}$ est négligeable pour les instances considérées dans la section expérimentale (moins de 150 tâches).

La borne supérieure $\overline{\mathcal{P}}_i$ est initialisée à l'ensemble des tâches càd $\{1, ..., n\}$.

Un raisonnement similaire est appliqué pour la variable ensembliste des successeurs $\mathcal{S}_i$ de l'objet i.

L'ensemble des prédécesseurs de l'objet i est utilisé pour filtrer la borne inférieure de B_i car nous savons que les objets dans $\mathcal{P}_i$ doivent être placés avant l'objet i. Pour un ensemble U comprenant des éléments de $\{1, ..., n\}$, nous notons $\text{sum}(U)$ le poids total des objets dans U, càd $\text{sum}(U) = \sum_{j \in U} s_j$.

Théorème 1 *Une contrainte redondante pour le BPCP est :*

$$\text{sum}(\mathcal{P}_i) = \sum_{k \leq B_i} L_k. \quad (6)$$

Preuve 1 *Les membres droit et gauche sont deux manières différentes de compter le poids cumulé des B_i premières bins. Le membre droit est la manière naturelle et le membre de gauche fait la somme du poids des objets se trouvant dans une bin précédente ou égale à B_i.* ■

Calcul de $\text{sum}(\mathcal{P}_i)$: Le calcul de $\text{sum}(\mathcal{P}_i)$ peut être réalisé à l'aide de variables binaires pour représenter les prédécesseurs mais une meilleure formulation en termes de filtrage et d'efficacité est possible à l'aide des variables ensemblistes.

En effet $\text{sum}(\mathcal{P}_i) = \sum_{j \in \mathcal{P}_j} s_j$ s'exprime tel quel dans Ilog Solver avec la contrainte globale `IloEqSum` réalisant une somme indicée par les éléments de la variable ensembliste. Cette contrainte globale prend en argument une fonction faisant la correspondance entre les indices des objets et leur poids : $f : \{1, ..., n\} \mapsto \{s_1, ..., s_n\} : f(j) = s_j$. La contrainte globale prend au total trois arguments : une variable ensembliste, une variable et une fonction :

$$\texttt{IloEqSum}(\mathcal{P}_i, \text{sum}(\mathcal{P}_i), f) \equiv \sum_{j \in \mathcal{P}_i} f(j) = \text{sum}(\mathcal{P}_i).$$

Calcul de $\sum_{k \leq B_i} L_k$: La formulation de $\sum_{k \leq B_i} L_k$ pourrait être réalisée à l'aide de m variables binaires pour chaque objet i spécifiant si une bin précède la bin dans laquelle se trouve l'objet i. Une meilleure formulation est possible en introduisant le tableau de m variables $\mathbf{CL} = [CL_1, ..., CL_m] : CL_i = \sum_{k=1}^{i} L_k$ pour $i \in [1, ..., m]$ (CL pour Cumulated Load càd poids cumulé). Avec ce tableau de variables, $\sum_{k \leq B_i} L_k$ s'écrit avec une contrainte element [16] comme CL_{B_i}.

En résumé, le modèle Ilog de la contrainte redondante (6) pour les prédécesseurs de l'objet i est :

$$\texttt{IloEqSum}(\mathcal{P}_i, \text{sum}(\mathcal{P}_i), f) \wedge \text{sum}(\mathcal{P}_i) = CL_{B_i}. \quad (7)$$

La contrainte (7) filtre les domaines de $\mathcal{P}_i$, B_i et de L_i's. Nous définissons des contraintes similaires pour les variables successeurs $\mathcal{S}_i$. Les résultats obtenus avec la formulation améliorée par les contraintes redondantes sont donnés dans la colonne D du Tableau 1. Les contraintes redondantes sont fortement bénéfiques pour le temps de calcul (1.07s contre 4.3s) et le nombre de backtracks (445 contre 2694).

5 Une contrainte globale pour le BPCP

La définition d'une contrainte globale BPCP est la suivante : $\texttt{BBCP}([B_1, ..., B_n], [L_1, ..., L_m], E)$ est satisfaite ssi

(i) $B_i \leq B_j, \forall (i, j) \in E$ et

(ii) $L_i = \sum_{\{j \in [1..n] \mid B_j = i\}} s_j$, $\forall i \in [1..m]$.

Les contraintes (i) et (ii) sont modélisées avec les contraintes (1-4), `IloPack` ainsi que les nouvelles contraintes redondantes composées de (5) et (7).

Pour cette contrainte globale, nous proposons également un nouvel algorithme avec une complexité de $O(n^2)$ pour filtrer d'avantage les domaines de $[B_1, ..., B_n]$ et $[L_1, ..., L_m]$. Ce filtrage n'inclut pas le filtrage obtenu avec les contraintes redondantes. C'est pourquoi il doit venir en plus du filtrage obtenu avec les contraintes redondantes.

Les contraintes redondantes (5) et (7) filtrent essentiellement la borne inférieure de B_i. En considérant le tableau des bornes supérieures des variables de poids des bins $[L_1^{\max}, ..., L_m^{\max}]$, les contraintes redondantes maintiennent :

$$B_i^{\min} \leftarrow \min\{j : \sum_{k=1}^{j} L_k^{\max} \geq \sum_{j \in \underline{\mathcal{P}}_i} s_j\} \quad (8)$$

La règle de filtrage (8) est un des filtrages réalisé par (7).

Exemple 1 *Un objet a un poids de 4 et trois prédécesseurs de poids 4,3,5. La hauteur de toutes les bins est 5. L'objet ne peut certainement pas être placé avant la bin 4 car pour la bin 4 nous avons $\sum_{k=1}^{4} L_k^{\max} = 20 \geq 16$ alors que pour la bin 3 nous avons $\sum_{k=1}^{3} L_k^{\max} = 15 < 16$.*

La règle (8) est une relaxation de la plus grande bornes inférieure pouvant être trouvée pour B_i :

- la règle fait l'hypothèse de préemption des objets c'est à dire qu'ils peuvent être coupés entre plusieurs bins, et
- la règle fait l'hypothèse que tous les prédécesseurs peuvent démarrer potentiellement à partir de la première bin.

Nous proposons un algorithme obtenant une meilleure borne inférieure qui conserve l'hypothèse de relâchement de préemption mais ne permettant plus à un prédécesseur j de commencer avant sa bin la plus tôt càd $B_j^{\min}$.

Notre algorithme nécessite que les prédécesseurs j soient triés de manière croissante selon leur bin la plus

tôt $B_j^{\min}$. Cela est réalisé en $\Theta(|\underline{\mathcal{P}}_i| + m)$ avec un l'algorithme *counting sort* [4] puisque les domaines des B_j prennent des valeurs dans $[1, ..., m]$. Cette complexité peut être simplifiée en $O(n)$ puisque $|\underline{\mathcal{P}}_i| < n$ et que typiquement $m \sim O(n)$ (moins de bins que d'objets).

L'algorithme 1 calcule la bin minimum dans laquelle l'objet i peut être placé en supposant que :

- chaque prédécesseur j ne peut pas commencer avant sa bin la plus tôt $B_j^{\min}$ mais peut finir dans n'importe quelle bin plus grande et
- un objet peut être divisé entre plusieurs bins (relaxation par préemption des objets).

L'algorithme 1 place d'abord les prédécesseurs de i c'est à dire les éléments de $\underline{\mathcal{P}}_i$. Cela est effectué dans la boucle externe **forall**. Ensuite, l'objet i est placé dans la bin la plutôt où il peut venir sans préemption. Nous supposons qu'il y a $m + 1$ bins de capacité $[L_1^{\max}, ..., L_m^{\max}, \sum_{i=1}^n s_i]$. La dernière bin est une bin fictive ayant une capacité suffisante pour accueillir tous les objets en même temps. Cela garanti la terminaison de la boucle **while**. La complexité de l'algorithme est de $O(|\underline{\mathcal{P}}_i| + m)$ où m est le nombre de bins. Pour n objets, la complexité devient $O(n^2)$.

L'algorithme 1 renvoie deux valeurs bin et $idle$. La valeur bin est utilisée pour filtrer la borne inférieure de B_i :

$$B_i^{\min} \leftarrow \max(B_i^{\min}, bin).$$

La valeur $idle$ est utilisée pour filtrer $L_{bin}^{\min}$. En effet, si B_i est affecté alors $bin = B_i$. Cela signifie que $L_{bin}^{\min}$ doit être au moins plus important que $L_{bin}^{\max} - idle$:

$$L_{bin}^{\min} \leftarrow \max(L_{bin}^{\min}, L_{bin}^{\max} - idle).$$

Nous utilisons évidemment un filtrage similaire utilisant la variable ensembliste $\mathcal{S}_i$ afin de filtrer la borne supérieure B_i.

Algorithme 1 : Considérant des bins de capacité maximum $[L_1^{\max}, ..., L_m^{\max}, \sum_{i=1}^n s_i]$, bin est l'index de bin le plus petit tel que tous les objets de l'ensemble $\underline{\mathcal{P}}_i$ ont été placés de manière préemptive dans une bin inférieure ou égale à bin sans débuter avant leur bin la plus tôt. La valeur $idle$ est la capacité restante dans cette bin.

```
bin ← 0
idle ← L_bin^max
forall j ∈ P_i \ {i} do
    /* invariant : bin est l'index de bin le plus petit tel que
       les objets {1, ..., j − 1} \ {i} ont tous été placés de
       manière préemptive dans une bin inférieure ou
       égale à bin mais sans débuter avant leur bin le plus
       tôt et idle est le poids libre restant dans cette bin. */
    if B_j^min > bin then
        bin ← B_j^min
        idle ← L_bin^max
    s ← s_j
    while s > 0 do
        if idle > s then
            idle ← idle − s
            s ← 0
        else
            s ← s − idle
            bin ← bin + 1
            idle ← L_bin^max
/* place l'objet i sans préemption */
if B_i^min > bin then
    bin ← B_i^min
    idle ← L_bin^max
while idle < s_i do
    bin ← bin + 1
    idle ← L_bin^max
idle ← idle − s_i
return bin, idle
```

6 Résultats expérimentaux

Nous proposons deux heuristiques différentes pour le PELAS-2. La première choisit la prochaine variable devant être instanciée sur base de la taille des domaines (*first-fail*) tandis que la seconde est basée sur la topologie du graphe de précédences et essaie de construire de manière heuristique une bonne solution qui satisfait les précédences. Pour chaque heuristique, les variables de décision à instancier sont $[B_1, ..., B_n]$ c'est à dire que pour chaque objet, nous devons décider la bin où il est placé.

- Heuristique 1 : La prochaine variable à instancier est celle avec le plus petit domaine (heuristique first-fail classique en PC). Pour briser les égalités, la priorité est donnée aux objets avec le plus gros poids. L'heuristique de valeur choisit de placer l'objet dans la bin la moins chargée couramment.
- Heuristique 2 : L'ordre d'instanciation des variables est statique : les variables sont instanciée dans un ordre topologique arbitraire du graphe de précédences (voir par exemple les nombres supérieurs dans les noeuds de la Figure 1). Ensuite, l'objet choisi est placé dans la première bin possible moins chargée que $\sum_{i=1}^n s_i/m$ (poids moyen des bins) ou dans la bin la moins chargée s'il n'y a pas de telle bin.

Nous avons sélectionné un sous ensemble d'instances du PELAS-2 du jeu de données de [13] possédant un nombre de tâches variant de 29 à 148. Le nom des instances est donné dans le Tableau 2 ainsi que le nombre de tâches indiqué entre parenthèses. Pour chaque graphe de précédences (instances) nous considérons trois problèmes respectivement avec 6, 10 et 14 stations de travail (bins).

Heuristique 1							Heuristique 2						
m	Time (s)			Objective			m	Time (s)			Objective		
	C	D	E	C	D	E		C	D	E	C	D	E
Buxey(29)							Buxey(29)						
6	1	**0**	**0**	55	55	55	6	3	**0**	**0**	55	55	55
10	**0**	1	**0**	34	34	34	10	300	**0**	**0**	34	34	34
14	7	**1**	2	25	25	25	14	0	0	0	25	25	25
Kilbrid(45)							Kilbrid(45)						
6	**1**	34	33	92	92	92	6	6	**0**	**0**	92	92	92
10	**6**	300	8	**56**	74	**56**	10	1	1	1	56	56	56
14	5	32	**2**	55	55	55	14	0	0	0	55	55	55
Hahn(53)							Hahn(53)						
6	**2**	**2**	**2**	2400	2400	2400	6	98	**1**	**1**	2400	2400	2400
10	3	**1**	**1**	1775	1775	1775	10	300	**1**	**1**	1827	**1775**	**1775**
14	25	**1**	**1**	1775	1775	1775	14	0	0	0	1775	1775	1775
Warnecke(58)							Warnecke(58)						
6	300	**1**	**1**	260	**258**	**258**	6	47	**1**	**1**	258	258	258
10	300	300	**71**	158	166	**155**	10	300	300	300	160	158	**156**$^{(155)}$
14	300	42	**19**	112	**111**	**111**	14	300	300	300	113	**112**	**112**$^{(111)}$
Tonge(70)							Tonge(70)						
6	**19**	68	68	585	585	585	6	126	**3**	**3**	585	585	585
10	300	300	300	355	**352**	**352**$^{(352)}$	10	300	**15**	**15**	369	**352**	**352**
14	300	300	300	326	**267**	**267**$^{(251)}$	14	300	161	**144**	260	**251**	**251**
Wee-mag(75)							Wee-mag(75)						
6	300	300	300	254	254	**252**$^{(250)}$	6	4	**2**	**2**	250	250	250
10	300	300	300	235	**201**	**201**$^{(150)}$	10	300	300	300	152	**151**	**151**$^{(150)}$
14	300	300	300	235	**201**	**201**$^{(108)}$	14	300	300	300	111	111	111$^{(108)}$
Lutz2(89)							Lutz2(89)						
6	300	**6**	**6**	142	**81**	**81**	6	300	**1**	**1**	83	**81**	**81**
10	300	300	**93**	142	62	**49**	10	300	**2**	**2**	51	**49**	**49**
14	300	300	300	348	**41**	**41**$^{(35)}$	14	300	**2**	**2**	36	**35**	**35**
Mukherje(94)							Mukherje(94)						
6	300	12	**10**	972	**704**	**704**	6	300	**5**	**5**	706	**704**	**704**
10	300	**43**	47	972	**424**	**424**	10	300	**5**	**5**	424	424	424
14	300	111	**93**	972	**311**	**311**	14	300	300	300	318	**313**	**313**$^{(311)}$
Barthold(148)							Barthold(148)						
6	**4**	5	5	939	939	**939**	6	300	300	300	940	940	940$^{(939)}$
10	300	**28**	**28**	574	**564**	**564**	10	300	300	300	566	566	566$^{(564)}$
14	300	300	300	569	**407**	**407**$^{(403)}$	14	300	**31**	34	404	**403**	**403**

TAB. 2 – Résultats obtenus avec les heuristiques 1 et 2 pour les configurations C, D et E.

Toutes les expériences ont été réalisées avec Ilog solver 6.3 et une machine possédant un CPU Intel® Xeon(TM) 2.80GHz. Le délai maximum pour la recherche a été arbitrairement fixé à 300 secondes.

Comme première expérimentation, nous proposons de résoudre le problème avec un Branch-and-Bound à l'aide des heuristiques 1 et 2 pour les trois différents modèles :

- C : modèle de PC état de l'art c'est à dire contraintes (1-4) and `IloPack`.
- D : C + les contraintes redondantes (5) et (7).
- E : D + le nouvel algorithme de filtrage. L'algorithme de filtrage de la contrainte globale est déclenché lorsqu'une des bornes d'une variable de poids des bins L_i change.

Les résultats obtenus (temps et meilleur objectif) pour les configurations C, D et E avec la première et la seconde heuristique sont donnés dans le Tableau 2. La valeur optimale est donnée en exposant entre parenthèses lorsque l'optimalité n'a pas pu être prouvé.

Analyse des résultats avec l'heuristique 1 : L'effet positif des contraintes redondantes et de la contrainte globale est assez flagrant. En effet, les configurations C, D et E permettent de résoudre et de prouver l'optimalité de respectivement 11, 17 et 20 instances sur un total de 27 endéans la limite de temps de 5 minutes (300 secondes). L'instance Lutz2 avec 10 stations de travail a pu être résolue uniquement à l'aide de la contrainte globale.

Analyse des résultats avec l'heuristique 2 : Les configurations C, D et E permettent de résoudre et prouver l'optimalité de respectivement 10, 20 et 20 instances sur 27 au total endéans la limite de temps de 5 minutes. L'effet positif des contraintes redondantes est toujours impressionnant mais la contrainte globale ne permet pas de résoudre plus d'instances. Pour l'instance Warnecke avec 10 stations, l'objectif atteint est meilleur avec la contrainte globale.

L'heuristique 2 permet de résoudre des instances qui ne sont pas solubles avec l'heuristique 1 (par exemple Barthold 14). Le contraire est également vrai puisque l'instance Warnecke avec 10 et 14 stations ne peut être résolue avec l'heuristique 2 mais est résolue aisément avec l'heuristique 1. Pour les instances les plus difficiles résolues par aucune des deux heuristiques (Wee-mag 10 and Wee-mag 14), l'heuristique 2 obtient de meilleures valeurs objectif ($151<201$ et $111<201$) très proches des valeurs optimales 150 et 108.

Comparaison avec l'algorithme dédié état de l'art : L'algorithme état de l'art pour ce problème est Salome 2 [8, 14]. Un fichier binaire de l'implémentation de l'algorithme est disponible sur *www.assembly-line-balancing.de*. Salome 2 trouve les solutions optimales de presque toutes les instances en moins d'une seconde. Néanmoins, comme nous, il ne parvient pas à trouver et prouver l'optimum pour les instances Wee-mag 10 et Wee-mag 14. Salome 2 utilise de nombreuses règles de dominances et de réductions spécifiques à ce problème et cette fonction objectif.

Problème généralisé d'équilibrage de lignes d'assemblage (PGELA) : PELAS-2 est un problème académique. Dans les véritables problèmes de lignes d'assemblage, des contraintes supplémentaires peuvent apparaître et dans ce cas, l'usage des règles de dominances dans Salome 2 n'est plus nécessairement valide. De possibles contraintes additionnelles sont [1] :

- certaines tâches doivent être assignées dans la même station,
- certaines tâches ne peuvent être assignées dans la même station,
- il y a une limite sur la valeur cumulée d'un attribut particulier des tâches,
- certaines tâches ne peuvent être assignées qu'à certaines stations,
- certaines tâches ne peuvent être assignées à une station particulière,
- certaines tâches nécessitent spécifiquement une station,
- certaines tâches doivent être séparées d'une distance minimum,
- certaines tâches doivent être séparées d'une distance maximum d'autres tâches.

Toutes ces contraintes additionnelles peuvent être ajoutées aisément dans notre modèle de PC sans rien changer d'autre alors que des algorithmes dédiés comme Salome 2 nécessiteraient probablement un profond remaniement pour gérer ces nouvelles contraintes. Afin de montrer la flexibilité de notre approche, nous avons ajouté quelques contraintes sur l'instance Barthold à 10 stations afin d'obtenir un problème généralisé d'équilibrage de ligne d'assemblage (PGELA) : $|B_{138} - B_{16}| \leq 2$, $|B_{104} - B_{41}| \geq 2$, $|B_{12} - B_{35}| \leq 2$, $|B_{65} - B_{76}| \leq 2$, $|B_{101} - B_{102}| \geq 2$, $|B_{83} - B_{113}| \geq 2$, $|B_{19} - B_{28}| \geq 3$ and $B_{16} = 4$.

Le temps nécessaire pour obtenir et prouver l'optimalité (662) pour les configurations C, D et E avec l'heuristique 1 sont respectivement 458, 171 et 143 secondes. L'heuristique 2 donne de mauvais résultats sur le PGELA car elle a été conçue spécifiquement pour le problème avec contraintes de précédences uniquement. Ici encore, les nouvelles contraintes redondantes et la contrainte globale aident vraiment à résoudre le problème plus rapidement. Ce problème ne peut être résolu avec l'algorithme dédié Salome 2 qui est actuellement l'état-de-l'art.

Un autre avantage de la programmation par contraintes est que la fonction objectif peut facilement être changée. Par exemple, il est souvent souhaitable de lisser les du-

rées de chaque station afin qu'elles soient le plus uniformes possible [1, 14, 10]. Ce problème est appelé le l'équilibrage de lignes vertical (Vertical Line Balancing). Il peut être traité efficacement en PC à l'aide des contraintes globales `spread` et `deviation` pour lisser les stations selon les critères de variance [9, 11] et de déviation absolue moyenne [12].

7 Conclusion and Perspectives

Nous proposons un modèle de PC pour le problème BBCP et le PELAS-2, permettant d'exprimer aisément de nouvelles contraintes telles que celles d'un PGELA. Nous proposons également de nouvelles contraintes redondantes et une nouvelle contrainte globale basées sur des variables ensemblistes représentant les prédécesseurs de chaque objet et dont la borne inférieure est initialisée à l'aide de la fermeture transitive sur le graphe de précédences.

Nous avons mené une étude expérimentale sur des jeux de données standards pour le PELAS-2, montrant la faisabilité, l'efficacité et la flexibilité de l'approche.

Dans le futur, nous souhaitons réaliser une hybridation de la PC et la recherche locale à l'aide d'une recherche à voisinage large. Nous aimerions également utiliser `spread` et `deviation` pour résoudre le problème vertical d'équilibrage de ligne car il n'existe à ce jour aucune méthode exacte pour ce problème.

Enfin, nous souhaitons améliorer le filtrage de notre contrainte globale à l'aide de bornes plus fortes telles que la borne de Martello et Toth.

Remerciements

Merci aux lecteurs anonymes pour leurs commentaires utiles et éclairants. Cette recherche est supportée par la région wallonne, le projet Transmaze (516207) et partiellement par le programme des pôles d'attraction interuniversitaires (État belge, Politique de science belge).

Références

[1] Christian Becker and Armin Scholl. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3) :694–715, 2006.

[2] Nils Boysen, Malte Fliedner, and Armin Scholl. A classification of assembly line balancing problems. *European Journal of Operational Research*, pages 674–693, 2007.

[3] C. Castro and S. Manzano. Variable and value ordering when solving balanced academic curriculum problem. *Proc. of the ERCIM WG on constraints*, 2001.

[4] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[5] C. Gervet. New structures of symbolic constraint objects : sets and graphs, 1993.

[6] B. Hnich, Z. Kiziltan, and T. Walsh. Modelling a balanced academic curriculum problem. *Proceedings of CP-AI-OR-2002*, 2002.

[7] ILOG-S.A. Ilog solver 6.3. user manual.

[8] Robert Klein and Armin Scholl. Maximizing the production rate in simple assembly line balancing : A branch and bound procedure. *European Journal of Operational Research*, 91(2) :367–385, 1996.

[9] Gilles Pesant and Jean-Charles Regin. Spread : A balancing constraint based on statistics. *Lecture Notes in Computer Science*, 3709 :460–474, 2005.

[10] B Rekiek, P De Lit, F Pellichero, E Falkenauer, and A Delchambre. Applying the equal piles problem to balance assembly lines. *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning*, pages 399–404, 1999.

[11] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. Simplification and extension of the spread constraint. *Third International Workshop on Constraint Propagation And Implementation*, 2006.

[12] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. The deviation constraint. *LNCS CP-AI-OR*, 4510 :269–284, 2007.

[13] Armin Scholl. Data of assembly line balancing problems. *Technische Universitat Darmstadt*, 93.

[14] Armin Scholl and Christian Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3) :666–693, 2006.

[15] Paul Shaw. A constraint for bin packing. In Mark Wallace, editor, *CP*, volume 3258 of *LNCS*, pages 648–662. Springer, 2004.

[16] Carillon J.-P. Van Hentenryck P. Generality versus specificity : an experience with ai and or techniques. *In Proceedings of AAAI-88*, 1988.

[17] W.-J. van Hoeve and I. Katriel. Global constraints. In F. Rossi, P. Van Beek, and T. Walsh, editors, *Handbook of constraint programming*, chapter 6, pages 169–208. Elsevier, 2006.