Contrainte de sac-à-dos à choix multiples dans les réseaux de fonctions de coûts

Pierre Montalbano[1*] **Simon de Givry**[1] **George Katsirelos**[2]

[1] Université Fédérale de Toulouse, ANITI, INRAE, UR 875, Toulouse, France
[2] Université Fédérale de Toulouse, ANITI, INRAE, MIA Paris, AgroParisTech,France
{pierre.montalbano,simon.de-givry}@inrae.fr gkatsi@gmail.com

Résumé

Les réseaux de contraintes pondérées (Cost Function Networks (CFNs)), aussi connus sous le nom de problèmes de satisfaction de contraintes pondérées (Weighted Constraint Satisfaction Problems (WCSPs)), peuvent représenter des fonctions décomposables de manière compacte, favorisant l'efficacité des algorithmes d'inférences. En particulier, la majorité des méthodes cherchant à calculer une borne duale (inférieur) pour minimiser un critère, passe par une solution faisable du dual de la relaxation linéaire. Ces méthodes se montrent plus efficaces que de résoudre la relaxation linéaire exactement, cependant elles sont exclusivement adaptées à la structure des relaxations linéaires des CFNs. Elles ne peuvent donc pas gérer les contraintes dont l'expression en extension est impossible, comme par exemple les contraintes linéaires de grande arité. Dans ce travail, nous montrons comment étendre les algorithmes de cohérences locales, de manière à pouvoir traiter des contraintes linéaires ainsi que des contraintes linéaires associées à des contraintes At Most One. Nous avons implémenté cet algorithme dans le solveur Toulbar2, un solveur exact de CFNs basé sur un algorithme de Séparation et Évaluation et sur la cohérence locale EDAC pour calculer les minorants pendant la recherche. Cela nous a permis de tester ce solveur sur différents problèmes qui jusqu'alors était hors de portée pour Toulbar2 comme les problèmes d'optimisation Pseudo-booléenne (PB) ou le problème du sac-à-dos avec un graphe de conflits (KPCG). Enfin nouc comparons nos résultats avec des solveurs PB et des solveurs de programmation linéaire en nombre entier, nous montrons des performances compétitives sur certaines familles des compétitions PB récentes et sur des instances KPCG. Nous améliorons également le calcul des séquences de solutions diverses pour le problème de conception de protéines (Computational Protein Design).

Abstract

Cost function networks (CFNs), also known as Weighted Constraint Satisfaction Problems, can compactly express large decomposable cost functions, which leads to efficient inference algorithms. In particular, most methods for computing dual (lower) bounds in minimization compute feasible dual solutions of a specific linear relaxation. These methods are more effective than solving the linear relaxation exacly, because its size is significantly larger than the original CFN. However, these algorithms are specialized to the structure of the linear relaxation of a CFN and cannot, for example, deal with constraints that cannot be expressed in extension, such as linear constraints of large arity.

In this work, we show how to extend soft local consistencies, a set of approximate inference techniques for CFNs, so that they handle linear constraints, as well as combinations of linear constraints with at-most-one constraints. We embedded the resulting algorithm in toulbar2, an exact Branch-and-Bound solver for CFNs which primarily relies on EDAC for dual bounds during search. This allows us to test this solver on problems on which it was previously not applicable, such as pseudo-Boolean optimization (PB) and knapsack with conflict graph (KPCG). Compared to dedicated PB solvers and integer linear programming solvers, we demonstrate state-of-the-art performance on some families from recent PB optimization evaluations and KPCG, and improve performance in computational protein design with diversity guarantees.

*Papier doctorant : Pierre Montalbano[1] est auteur principal.

1 Contexte

Definition 1. *Un problème de satisfaction de contraintes pondérées (WCSP) est un tuple* $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C}, \top)$ *où* $\boldsymbol{X}$ *est un ensemble de variables discrètes, de domaine* $\boldsymbol{D}(x)$ *pour* $x \in \boldsymbol{X}$. $\boldsymbol{C}$ *est un ensemble de fonctions de coûts, de portée* $scope(c) \subseteq \boldsymbol{X}$ *pour* $c \in \boldsymbol{C}$. $\top$ *définit un coût maximal indiquant une*

affectation interdite.

La taille de la portée est appelée arité. Une affectation partielle notée τ est une affectation de toutes les variables $x_i \in scope(\tau)$ à une valeur de leur domaine $\boldsymbol{D}(x_i)$. L'ensemble des affectations partielles d'une portée $\boldsymbol{S}$ est noté $\tau(\boldsymbol{S})$. L'évaluation du coût d'une affectation partielle τ par une contrainte $c_{\boldsymbol{S}}$ est notée $c(\tau|_{\boldsymbol{S}})$ avec $\boldsymbol{S} \subseteq scope(\tau)$. Dans le cadre du WCSP tous les coûts considérés sont des entiers positifs bornés par $\top$, de ce fait si $c(\tau|_{\boldsymbol{S}}) = \top$ alors l'affectation τ ne constitue pas une solution faisable. Une contrainte $c_{\boldsymbol{S}}$ est dite dure si pour toute affectation partielle τ, $c_{\boldsymbol{S}}(\tau) \in \{0, \top\}$, sinon on dit que la contrainte est souple. Un WCSP composé uniquement de contraintes dures est un CSP. Dans la suite les termes *fonction de coûts* et *contrainte* sont équivalents. Une affectation complète est donnée par $scope(\tau) = \boldsymbol{X}$ et le coût d'une affectation complète τ est donné par $c(\tau) = \sum_{c_{\boldsymbol{S}} \in \boldsymbol{C}} c(T|_{\boldsymbol{S}})$. Résoudre un WCSP revient à trouver l'affectation complète minimisant $c(\tau)$. Ce problème est défini comme NP-dur [32].

Les WCSP sont habituellement résolus avec un algorithme de Séparation et Évaluation (Brand-and-Bound ($B\&B$)). A chaque noeud de l'arbre issu du $B\&B$, le solveur calcule une borne et ferme le noeud si cette borne est plus grande que la solution courante ou si le noeud définit un problème infaisable. Les algorithmes présentés ici utilisent la fonction de coût $c_\varnothing$ pour représenter un minorant naturel du WCSP et cherchent une re-paramétrisation du problème augmentant $c_\varnothing$. Une re-paramétrisation P' d'un WCSP P est un WCSP de structure identique (contraintes de même portée) mais dont les coûts peuvent localement être différents, cependant $c_P(\tau) = c_{P'}(\tau)$ pour toute affectation complète τ. Il sera possible d'augmenter le minorant $c_\varnothing$ s'il existe une fonction de coûts $c_{\boldsymbol{S}}$ telle que pour tout $\tau \in \tau(\boldsymbol{S})$, $c_{\boldsymbol{S}}(\tau) > 0$. Toutes les re-paramétrisations considérées sont obtenues par une séquence de transformations préservant l'équivalence (en anglais, *Equivalence Preserving Transformation* (EPT)) menant à l'augmentation de $c_\varnothing$. Soit deux fonctions de coûts $c_{\boldsymbol{S}_1}, c_{\boldsymbol{S}_2}$ avec $\boldsymbol{S}_1 \subset \boldsymbol{S}_2$. La procédure MoveCost décrit comment déplacer les coûts entre ces 2 fonctions de coûts. On peut observer que si τ est utilisé dans une affectation complète, alors il y a exactement une extension de τ vers $\boldsymbol{S}_2$. De ce fait le coût de l'affectation complète est inchangé indépendemment de l'attribution du coût α à τ ou à toutes les extensions τ' dans $\boldsymbol{S}_2$. Dans la suite si α est positif, le mouvement s'opère d'une fonction d'arité plus grande vers une fonction d'arité plus faible et cette opération est appelée projection, notée $project(c_{\boldsymbol{S}_1}, c_{\boldsymbol{S}_2}, \tau_1, \alpha)$. Sinon si α est négatif, le coût se déplace vers une fonction d'arité plus grande et on appelle cette opération extension, notée $extend(c_{\boldsymbol{S}_1}\tau_1, c_{\boldsymbol{S}_2}, -\alpha)$. Le mouvement de $|\boldsymbol{S}_2| = 1$ vers $\boldsymbol{S}_1 = \varnothing$ est une projection unaire, notée $unaryProject(c_i, v, \alpha)$. Nous n'effectuons pas d'extension à partir de $c_\varnothing$, celui-ci ne pourra pas diminuer au cours de la recherche.

Procedure MoveCost($\boldsymbol{S}1, \boldsymbol{S}_2, \tau, \alpha$) : Déplace un coût α entre le tuple τ de la portée $\boldsymbol{S}_1$ et les tuples étendus de τ dans la portée $\boldsymbol{S}_2$

Data: Scopes $\boldsymbol{S}_1 \subset \boldsymbol{S}_2$
Data: $\tau \in \tau(\boldsymbol{S}_1)$
Data: cost α to move

$c(\tau|_{\boldsymbol{S}_1}) \leftarrow c(\tau|_{\boldsymbol{S}_1}) + \alpha$;
foreach $\tau' \in \tau(\boldsymbol{S}_2) \mid \tau'|_{\boldsymbol{S}_1} = \tau$ **do**
  $c(\tau|_{\boldsymbol{S}_2}) \leftarrow c(\tau|_{\boldsymbol{S}_2}) - \alpha$;

L'objectif est donc de trouver une séquence d'EPTs menant à l'augmentation de $c_\varnothing$, cependant trouver les opérations menant à une re-paramétrisation optimale n'est pas évident. Il a été prouvé qu'elle peut être obtenue à partir d'une solution duale optimale de la relaxation linéaire du WCSP [11], appelée *polytope local*.

$$\min \sum_{c_{\boldsymbol{S}} \in \boldsymbol{C}, \tau \in \tau(\boldsymbol{S})} c(\tau|_{\boldsymbol{S}}) \times y_\tau$$

$$s.t.$$

$$y_\tau = \sum_{\tau' \in \tau(\boldsymbol{S}_2), \tau'|_{\boldsymbol{S}_1} = \tau} y_{\tau'} \quad \forall c_{\boldsymbol{S}_1}, c_{\boldsymbol{S}_2} \in \boldsymbol{C} | \boldsymbol{S}_1 \subset \boldsymbol{S}_2, \tau \in \tau(\boldsymbol{S}_1), |\boldsymbol{S}_1| \geq 1$$

$$\sum_{\tau \in \tau(\boldsymbol{S})} y_\tau = 1 \qquad \forall c_{\boldsymbol{S}} \in \boldsymbol{C}, |\boldsymbol{S}| \geq 1$$

Cependant résoudre ce problème à l'optimalité est souvent très coûteux car même si l'on considère un WCSP uniquement composé de contraintes binaires, il y a $O(ed^2 + nd)$ variables et $O(end^2)$ valeurs non-nulles dans sa matrice, où e est le nombre de fonctions de coûts, n le nombre de variables et d la taille des domaines. De plus la structure du problème ne permet pas de définir un algorithme spécifiquement efficace, en effet il a été prouvé qu'il est aussi difficile de résoudre ce problème que de résoudre un problème linéaire arbitraire [31]. De ce fait en pratique les recherches se sont dirigées vers le calcul d'une solution faisable du problème dual. Différents algorithmes ont ainsi été proposé, certains issus du domaine du traitement d'image [22, 40, 38, 23, 37] ou encore de la programmation par contraintes [25, 11], ces derniers sont appelés *algorithmes de cohérences locales souples*. Nous ne détaillons pas précisément tous les algorithmes de cohé-

rences locales existants mais nous supposons que le WCSP vérifie la propriété suivante :

Definition 2. *Un WCSP est Node Consistent (NC) [25] si pour tout* $c_{\boldsymbol{S}} \in \boldsymbol{C}$ *avec* $|\boldsymbol{S}| = 1$ *il existe un* $\tau \in \tau(c_{\boldsymbol{S}})$ *tel que* $c_{\boldsymbol{S}}(\tau) = 0$ *et pour tout* $\tau \in \tau(c_{\boldsymbol{S}})$, $c_\varnothing + c(\tau|_{\boldsymbol{S}}) < \top$.

Definition 3. *Un WCSP est* $\emptyset$*-Inverse Consistent (*$\emptyset$*IC) [41] si pour tout* $c_{\boldsymbol{S}} \in \boldsymbol{C}$ *il existe* $\tau \in \tau(\boldsymbol{S})$ *tel que* $c(\tau|_{\boldsymbol{S}}) = 0$.

Definition 4. *Un WCSP est Existential Arc Consistent (EAC) [12] s'il est NC et si pour toute variable* $x_i \in \boldsymbol{X}$ *il existe une valeur* $v \in \boldsymbol{D}_{x_i}$ *telle que pour toute contrainte* $c_{\boldsymbol{S}} \in \boldsymbol{C}, |\boldsymbol{S}| > 1$, *il existe une affectation partielle* τ *vérifiant* $\tau|_{x_i} = v$ *et* $\sum_{x_j \in S} c_j(\tau|_{x_j}) + c(\tau|_{\boldsymbol{S}}) = 0$. *La valeur* v *est appelée support EAC.*

Cette dernière définition s'applique uniquement aux fonctions de coûts binaires [1]. Une version plus faible pouvant être appliquée aux fonctions de coûts globales est introduite dans [28] et permet notamment d'éviter le problème d'oscillation des coûts. Nous proposons ici une autre approche plus faible que EAC en nous fondant sur $\emptyset$IC. Nous renforçons la définition précédente en prenant en compte les coûts unaires.

Definition 5. *Un WCSP est Full* $\emptyset$*-Inverse Consistent (F*$\emptyset$*IC) si pour tout* $c_{\boldsymbol{S}} \in \boldsymbol{C}$ *il existe* $\tau \in \tau(\boldsymbol{S})$ *tel que* $\sum_{x_j \in S} c_j(\tau|_{x_j}) + c(\tau|_{\boldsymbol{S}}) = 0$.

F$\emptyset$IC est plus faible que T-DAC [2]. C'est aussi plus faible que EAC sur les réseaux de contraintes binaires, et incomparable avec EAC faible [28]. Il existe des algorithmes permettant d'établir les cohérences définies précédemment, ils sont basés sur une solution duale faisable du polytope locale. Cependant ils requièrent d'exprimer toutes les contraintes en extension [2], ce qui est impossible pour un grand nombre de contraintes classiques en programmation par contraintes. Ces contraintes dites *globales* peuvent impliquer un nombre arbitraire de variables, il est donc nécessaire de définir des algorithmes sur-mesure pour établir une cohérence locale souple sur un problème impliquant des contraintes globales. En plus de supprimer les valeurs n'apparaissant dans aucune solution faisable, ces algorithmes cherchent une solution optimale en utilisant les coûts unaires et une (parfois plusieurs) contraintes globales. Puis ils exploitent ce coût optimal en re-paramétrisant le problème.

Ici, nous traitons les contraintes Pseudo-Booléennes (PB). Ce sont des contraintes de la forme $\sum_{x_i \in S} w_i x_i \geq C$, où S est une portée, $x_i \in S$ sont des variables Booléennes, w_i et C sont des constantes. Les contraintes avec un opérateur $\leq$ peuvent être transformées en appliquant un facteur -1 de chaque coté de la contrainte.

Il est possible de détecter en un temps linéaire s'il existe au moins une solution, en testant si $\sum_{x_i \in S} \max(0, w_i) \geq C$. De même il est possible de vérifier que chaque valeur apparaît dans au moins une solution.

Une contrainte at-most-one (AMO) est une contrainte de la forme $\sum_{x_i \in S} x_i \leq 1$, ou $\sum_{x_i \in S} -x_i \geq -1$. C'est une contrainte exactly-one (EO) si $\sum_{x_i \in S} x_i = 1$.

2 Travaux connexes

Les problèmes définis par des contraintes linéaires pseudo-Booléennes, sont une généralisation du problème SAT. Les solveurs PB SAT utilisent des techniques d'apprentissages [29], soit en traduisant directement le problème au format CNF [17, 35], soit en généralisant le mécanisme d'apprentissage de clauses aux contraintes PBs[14, 18]. Ces solveurs ne calculent pas de minorant durant la recherche et reposent en grande partie sur l'analyse des conflits pour prouver l'optimalité du problème. Le travail de Devriendt et al [20] est une exception, en effet ils utilisent un solveur LP pour obtenir des bornes durant la recherche et apprendre des contraintes à partir d'une violation d'une borne. Ils limitent cependant le nombre d'itération du solveur LP afin de garder un temps de recherche raisonnable. C'est une différence majeure avec notre approche qui utilise une solution sub-optimale mais sans limiter les ressources déployées pour l'obtenir. Les solveurs PB peuvent aussi exploiter la présence de contraintes AMO ou EO pour renforcer la propagation des contraintes PB [5, 7].

La relaxation linéaire du polytope local permet d'utiliser les solveurs de programmation linéaire en nombre entier (Integer Linear Programming (ILP)) pour résoudre les WCSPs. De plus utilisant une méthode de résolution générale ils n'ont pas de difficulté à prendre en compte les contraintes linéaires ou des combinaisons de contraintes linéaires supplémentaires. Cependant même pour des implémentations hautement optimisées, la taille du polytope locale est trop large [19] pour une résolution efficace. En pratique les solveurs spécialisés en WCSP sont plus appropriés.

Du coté WCSP, un travail similaire a été effectué sur un cas particulier de contrainte PB appelé contrainte de clique [13], l'approche présentée ici généralise ce cas. Dlask et Werner [15, 16] ont montré comment gérer des LPs en utilisant un algorithme de descente de coordonées en bloc (BCD) et une généralisation

1. Une extension aux coûts ternaires a été définie dans [36].
2. Ou bien d'avoir un algorithme polynomial pour déterminer le tuple de coût minimum $\sum_{x_j \in S} c_j(\tau|_{x_j}) + c(\tau|_{\boldsymbol{S}})$ [2].

de VAC. Cependant malgré des avancées récentes [39], cette méthode est trop coûteuse pour être utilisée à tous les noeuds du solveur B&B.

Lee et Shum [26] ont montré que certaines contraintes globales peuvent être décrites par un ensemble de contraintes linéaires souples, cependant cela nécessite l'utilisation d'un solveur LP. Enfin il est possible de décomposer les contraintes linéaires en utilisant des fonctions de coûts d'arité 3 et des variables intermédiaires [3]. Cependant la taille des domaines des variables intermédiaires augmente linéairement avec les coefficients de la contrainte PB.

3 Contrainte pseudo-Booléenne dans le cadre WCSP

La contrainte que nous considérons ici est $\sum_{z_i \in S} w_i y_i \geq C$, associée à des contraintes AMO de portée P_i où $P_1, \ldots, P_k$ est un partitionnement des variables. De manière équivalente, soit $\boldsymbol{S}$ un ensemble de variables d'un WCSP de domaine arbitraire, et w_{iv} le poids associé à chaque valeur. Nous considérons la contrainte : $\sum_{x_i \in S, v \in \boldsymbol{D}(x_i)} w_{iv} x_{iv} \geq C$, où x_{iv} est la variable 0/1 qui prend la valeur 1 si $x_i = v$. Les variables du WCSP ne pouvant être affectées qu'à une seule valeur, elles définissent implicitement un partitionnement des variables 0/1. De plus il est possible d'associer une contrainte EO à chaque partition, en effet ajouter une variable 0/1 avec un poids nul permet de transformer une contrainte AMO en contrainte EO.

Nous voulons établir un niveau de cohérence équivalent à $F\emptyset IC$. Cependant c'est un problème NP-difficile, en effet minimiser la somme des coûts unaires tout en cherchant à vérifier une contrainte linéaire définit le problème du sac-à-dos (avec les signes inversés). De ce fait, nous n'établissons pas complètement $F\emptyset IC$, nous détectons simplement certains cas où la contrainte n'est pas $F\emptyset IC$ et procédons à des mouvements de coûts. Nous conservons une trace des différents mouvements de coûts en associant chaque variable x_{iv} à une quantité δ_{iv} et $c_\varnothing$ à $\delta_\varnothing$. Si l'EPT $project(c_i, c_{PB}, \{x_i = v\}, \alpha)$ est effectuée alors $\delta_{iv} \rightarrow \delta_{iv} - \alpha$, si l'EPT $extend(c_i, \{x_i = v\}, c_{PB}, \alpha)$ est effectuée alors $\delta_{iv} \rightarrow \delta_{iv} + \alpha$. Initialement $\delta_\varnothing = 0$ et $\delta_{iv} = 0$ pour tout i, v.

$$\min \sum_i \sum_v \delta_{iv} x_{iv} - \delta_0 \tag{1}$$

$$s.t.$$

$$\sum_i \sum_v w_{iv} x_{iv} \geq C \tag{2}$$

$$\sum_v x_{iv} \leq 1, \quad \forall i \tag{3}$$

$$x_{iv} \in \{0, 1\}, \quad \forall i \tag{4}$$

Nous appelons cette formulation $ILP_\emptyset$. A tout moment de la recherche les coûts δ permettent d'évaluer le coût d'un tuple sur la contrainte linéaire. De ce fait si $opt(ILP_\emptyset) > 0$, la contrainte n'est pas $\emptyset IC$ et il est possible d'augmenter $c_\varnothing$. Cependant, pour l'établissement de $F\emptyset IC$ il est nécessaire de prendre en compte les coûts unaires. De ce fait, bien que $ILP_\emptyset$ représente la contrainte, notre propagateur va s'intéresser à une version modifiée.

$$\min \sum_i \sum_v (\delta_{iv} + c_i(v)) x_{iv} - \delta_0 \tag{5}$$

Nous appelons ce problème $ILP_{F\emptyset}$. La contrainte est $F\emptyset IC$ si $opt(F\emptyset IC) = 0$. Dans la suite nous utilisons $p_{iv} = \delta_{iv} + c_i(v)$.

Les 2 problèmes $ILP_\emptyset$ et $ILP_{F\emptyset}$ étant NP-difficiles, nous relâchons la contrainte d'intégrité (4) : $0 \leq x_{iv} \leq 1$ et résolvons les programmes linéaire obtenus, respectivement $LP_\emptyset$ et $LP_{F\emptyset}$. Avec cette relaxation, nous savons que si $opt(LP_{F\emptyset}) > 0$ alors la contrainte n'est pas $F\emptyset IC$, de même avec $LP_\emptyset$ et $\emptyset IC$.

$LP_{F\emptyset}$ est un problème équivalent (exprimé en minimisation au lieu de maximisation) au problème de sac à dos à choix multiples (MCKP, [30]), ou encore du *knapsack problem with special ordered sets* [21].

3.1 Problème dual et solution optimale

On obtient une solution optimale $\boldsymbol{x}^*$ du problème $LP_{F\emptyset}$ en appliquant l'algorithme décrit par Pisinger [30]. Cet algorithme donne la solution optimale $\boldsymbol{x}^*$ en $O(n \log n)$ où n est le nombre de variables 0/1. De plus soit $\boldsymbol{x}^*$ est une solution entière soit il contient exactement deux valeurs fractionnelles, dans ce cas la variable $x_k \in \boldsymbol{X}$ vérifiant $\exists (v, v')$ tel que $0 < x^*_{kv}, x^*_{kv'} < 1$ se nomme *split class* et x^*_{kv}, x^*_{kv+1} sont les *splits variables*. Nous notons par o le coût optimal de la solution $\boldsymbol{x}^*$ $o = \sum_i \sum_v p_{iv} x^*_{iv}$. Le début de l'exemple 1 illustre l'utilisation de l'algorithme de Pisinger.

A présent, si l'on regarde le problème dual de $LP_{F\emptyset}$:

$$\begin{aligned} \max C \times y_{cc} + \sum_i y_i \quad & (6) \\ s.t. & \\ \forall i, v \quad & y_{cc} \times w_{iv} + y_i \leq p_{iv} \\ y_{cc} \geq 0 & \end{aligned}$$

Où y_{cc} est la variable duale correspondant à la contrainte de capacité et y_i correspond à la contrainte EO de x_i. En connaissant la solution optimale primale, il est aisé de calculer la solution optimale duale. Soit x_k la split class, $x_{ks}, x_{ks'}$ les splits variables et pour $i \neq k$, définissons la variable x_{is} comme la variable utilisée dans la solution optimale, *i.e.*, $x^*_{is} = 1$.

- $y_{cc} = \frac{p_{ks} - p_{ks'}}{w_{ks} - w_{ks'}}$
- $y_k = p_{ks} - y_{cc} \times w_{ks} = p_{ks'} - y_{cc} \times w_{ks'}$
- Pour $i \neq k$, $y_i = p_{is} - y_{cc} \times w_{is}$

La solution optimale duale va nous permettre de calculer les coûts réduits de chacune des variables x_{iv}.

Definition 6. *Le coût réduit d'une variable x sous une solution duale $\mathbf{y}$ est le slack de la contrainte duale correspondant à x, noté $cr^{\mathbf{y}}(x_{iv})$ ou $cr(x_{iv})$ si la solution duale est implicite.*

Le coût réduit de x peut être interprété comme le montant dont on doit diminuer le coefficient de x dans la fonction objective afin d'avoir $x > 0$ dans la solution optimale. Ces coûts vont nous permettre de définir notre re-paramétrisation. Dans le cas spécifique de $LP_{F\emptyset}$ nous avons :

- $\forall i,\ cr(x_{is}) = 0$
- $cr(x_{ks}) = cr(x_{ks'}) = 0$
- $\forall v \neq s : cr(x_{iv}) = p_{iv} - y_{cc} \times w_{iv} - y_i$

Observation 1. *Le problème $LP'_{F\emptyset}$ identique à $LP_{F\emptyset}$ mais avec $p'_{iv} = p_{iv} - cr(x_{iv})$ vérifie $opt(LP'_{F\emptyset}) = opt(LP_{F\emptyset})$.*

Démonstration. Les coefficients des variables avec une valeur > 0 dans $\mathbf{x}^*$ ont les mêmes coefficients dans $LP_{F\emptyset}$ et $LP'_{F\emptyset}$. De ce fait la solution $\mathbf{x}^*$ a pour coût o dans $LP_{F\emptyset}$ et $LP'_{F\emptyset}$. La solution duale optimale $\mathbf{y}^*$ reste faisable dans $LP'_{F\emptyset}$, de plus la fonction objective du problème dual n'ayant pas changée $\mathbf{y}^*$ a un coût o. Donc $opt(LP'_{F\emptyset}) = o = opt(LP_{F\emptyset})$. □

L'exemple 1 illustre l'obtention des coûts réduits à partir d'une solution duale optimale.

Exemple 1. *Considérons le problème de sac à dos à choix multiples suivant :*

$$\begin{aligned} \min 40x_{11} + 55x_{12} + 85x_{13} + 47x_{21} + 95x_{22} & \\ s.t. & \\ 4x_{11} + 14x_{12} + 24x_{13} + 16x_{21} + 40x_{22} >= 40 & \\ \forall i, \sum_v x_{iv} = 1 & \\ 0 \leq x_{ij} \leq 1 & \end{aligned}$$

L'algorithme de Pisinger donne la solution optimale $x^* = \{0, 1, 0, \frac{7}{12}, \frac{5}{12}\}$ avec un coût $o = 55 + \frac{7}{12} \times 47 + \frac{5}{12} \times 95 = 122$.

Nous déduisons la solution optimale duale suivante : $y_{cc} = 2, y_1 = 55 - 2 \times 14 = 27, y_2 = 47 - 2 \times 16 = 15$ on observe bien que $40y_{cc} + y_1 + y_2 = 122$.

Nous obtenons les coûts réduits suivants : $cr(x_{12}) = cr(x_{21}) = cr(x_{22}) = 0$ et $cr(x_{11}) = 40 - 2 \times 4 - 27 = 5$, $cr(x_{13}) = 85 - 2 \times 24 - 27 = 10$, nous en déduisons que remplacer la fonction objective par la fonction objective suivante ne change pas le coût de la solution optimale :

$$\min 35x_{11} + 55x_{12} + 75x_{13} + 47x_{21} + 95x_{22}$$

On observe que la solution $\boldsymbol{x^*} = \{0, 1, 0, \frac{7}{12}, \frac{5}{12}\}$ est toujours optimale.

3.2 Propagateur de contrainte pseudo-Booléenne

Nous allons prouver ici qu'il est possible d'augmenter le minorant d'au moins o (le coût de la solution optimale $\boldsymbol{x^*}$). Notre objectif est d'étendre le moins de coûts unaires possible tout en vérifiant que $opt(LP_\emptyset) = o = opt(LP_{F\emptyset})$ afin de pouvoir projeter o sur $c_\varnothing$. La séquence d'EPTs permettant d'obtenir une augmentation de $c_\varnothing$ est obtenue grâce aux coûts réduits.

Si l'on étend ou projette la quantité $|c_i(v) - cr(x_{iv})|$ entre les coûts unaires et la contrainte, on remarque que pour les variables vérifiant $x^*_{iv} > 0$ on a $cr(x_{iv}) = 0$, on étend donc la totalité du coût unaire $c_i(v)$. Ce qui signifie que l'on a $\sum_i \sum_v \delta_{iv} x^*_{iv} = o$. Pour tout i, v, $|c_i(v) - cr(x_{iv})| = |y_i + y_{cc} \times w_{iv}) - \delta_{iv}|$, nous obtenons donc les nouveaux coûts par les opérations décrites dans l'algorithme TransformPB.

Theorem 7. *L'algorithme TransformPB définit une séquence de transformations préservant l'équivalence.*

Démonstration. Rappelons que $p_{iv} = c_i(v) + \delta_{iv}$, et $cr(x_{iv}) \geq 0$. Si $|c_i(v) - cr(x_{iv}) \geq 0$ alors l'opération $extend(c_i, \{x_i = v\}, c_{PB}, c_{iv} - cr(x_{iv}))$ est valide.

Si $c_i(v) - cr(x_{iv}) < 0$ alors cela signifie que le coût de toutes solutions x' avec $x'_{iv} = 1$ est au moins $o - c_i(v) + cr(x_{iv})$ donc l'opération $project(c_i, c_{PB}, \{x_i = v\}, -c_i(v) + cr(x_{iv}))$ est valide.

Procedure TransformPB(c_{PB}, y_{cc}, y_i, o)

Data: c_{PB} : une contrainte
Data: y_{cc}, y_i, o : une solution optimale duale $LP_{F\emptyset}$

for *all the variables* x_{iv} **do**
  $c_i(v) \leftarrow c_i(v) - y_{cc} \times w_{iv} - y_i + \delta_{iv}$;
  $\delta_{iv} \leftarrow y_{cc} \times w_{iv} + y_i$;
$c_\varnothing \leftarrow c_\varnothing + o$;
$\delta_\varnothing \leftarrow \delta_\varnothing + o$;

Finalement pour vérifier que la séquence d'EPT justifie une augmentation de $c_\varnothing$ par o, nous calculons l'optimum de $LP_\emptyset$. D'après l'observation 3.1, $opt(LP_\emptyset) = o$, nous pouvons donc projeter o sur $c_\varnothing$ et augmenter $\delta_\varnothing$ pour obtenir $opt(LP_\emptyset) = opt(LP_{F\emptyset}) = 0$. □

Nous pouvons affiner le raisonnement en nous approchant de la solution optimale entière. Pour cela nous augmentons $c_\varnothing$ par l'arrondi supérieur de o, dans ce cas il faut également prendre l'arrondi supérieur pour tous les mouvements de coûts effectués. L'utilisation des arrondis ne nous permet plus d'utiliser l'observation 3.1, mais nous vérifions toujours que $opt(LP_\emptyset) = 0$.
Notre algorithme est donné par la procédure Propagate. Nous débutons par établir une cohérence des domaines sur la contrainte PB, puis résolvons $LP_{F\emptyset}$ à l'optimalité. S'il y a plusieurs solutions nous préférons celle minimisant les coûts réduits associés au support EAC de chaque variable. Pour finir nous utilisons la procédure TransformPB. Bien que $LP_\emptyset$ est utile pour expliquer l'algorithme, il n'est jamais résolu durant la recherche.

Theorem 8. *La complexité de la procédure Propagate est $O(nd \log(nd))$ où n est le nombre de variables WCSP impliqué dans la contrainte et d la taille maximale des domaines. (Notre problème a donc nd variables booléennes).*

Démonstration. L'opération la plus coûteuse est l'application de l'algorithme de Pisinger, qui a une complexité de $O(N \log N)$, où N est le nombre de variables. Dans notre cas $N = nd$, donc l'algorithme de Pisinger est en $O(nd \log nd)$. La cohérence de domaine comme dit en introduction peut se faire en temps linéaire. Enfin, la procédure TransformPB boucle une seule fois sur toutes les variables binaires, et effectue des opérations en temps constant. De ce fait la complexité totale est $O(nd \log nd)$. □

Procedure Propagate(c_{PB})

Data: c_{PB} : une contrainte pseudo-Booléenne avec une partition en contraintes EO
DomainConsistency(c_{PB}) ;
(y_{cc}, y_i, o) = DualSolve($LP_{F\emptyset}$) ;
TransformPB(c_{PB}, y_{cc}, y_i, o) ;

Exemple 2. *En reprenant l'exemple utilisé dans la section 4.1, nous avions les coûts réduits suivants : $cr(x_{12}) = cr(x_{21}) = cr(x_{22}) = 0$, $cr(x_{11}) = 5$, $cr(x_{13}) = 10$, le coût optimal était 122. On en déduit les mouvements de coûts suivants :*
$c_1(1) \leftarrow c_1(1) - 35 = 5$, *et* $\delta_{11} \leftarrow 35$
$c_1(2) \leftarrow c_1(2) - 55 = 0$, *et* $\delta_{12} \leftarrow 55$
$c_1(3) \leftarrow c_1(3) - 75 = 10$, *et* $\delta_{13} \leftarrow 75$
$c_2(1) \leftarrow c_2(1) - 47 = 0$, *et* $\delta_{21} \leftarrow 47$
$c_2(2) \leftarrow c_2(2) - 95 = 0$, *et* $\delta_{22} \leftarrow 95$
$c_\varnothing \leftarrow o = 122$
$\delta_\varnothing \leftarrow 122$
On remarque bien : $\frac{\delta_{12}-\delta_{11}}{w_{12}-w_{11}} = \frac{20}{10} = 2 = \frac{\delta_{13}-\delta_{12}}{w_{13}-w_{12}} = \frac{\delta_{22}-\delta_{21}}{w_{22}-w_{21}}$. *Si l'on construit la table des affections possibles pour le problème $LP_\emptyset$ obtenu après les extensions on observe :* $c(x_1 = 1, x_2 = 2) = 8$, $c_(x_1 = 2, x_2 = 2) = 28$, $c_(x_1 = 3, x_2 = 1) = 0$, $c_(x_1 = 3, x_2 = 2) = 48$ *les autres affectations ne vérifiant pas la contrainte.*
On observe que la solution optimale est 0, nos extensions justifient bien l'augmentation de $c_\varnothing$. Maintenant supposons que des opérations extérieures à la contrainte aient modifié les coûts unaires : $c_1(1) \rightarrow c_1(1)+16 = 21$, $c_1(2) \rightarrow c_1(2) + 30 = 30$, $c_1(3) \rightarrow c_1(3) - 9 = 1$. *Nous souhaitons calculer un nouveau minorant pour la contrainte PB en résolvant le problème $LP_{F\emptyset}$: :*

$$\begin{aligned} \min(21+35)x_{11} + (30+55)x_{12} + (1+75)x_{13} + 47x_{21} \\ +95x_{22} - 122 \\ s.t. \\ 4x_{11} + 14x_{12} + 24x_{13} + 16x_{21} + 40x_{22} >= 40 \\ \forall i, \sum_v x_{iv} = 1 \\ 0 \leq x_{ij} \leq 1 \end{aligned}$$

La solution optimale est $x^* = \{0, 0, 1, 1, 0\}$ et son coût est $o = 76 + 47 - 122 = 1$. On déduit la solution optimale duale suivante : $y_{cc} = e_{113} = 1, y_1 = 76 - 24 = 52, y_2 = 47 - 16 = 31$ et les coûts réduits sont $cr(x_{13}) = cr(x_{21}) = cr(x_{11}) = 0$ et $cr(x_{12}) = 19$, $cr(x_{22}) = 24$. On effectue les mouvements de coûts suivants :
$c_1(1) \leftarrow c_1(1) - 56 + 35 = 0$, *et* $\delta_{11} \leftarrow 56$
$c_1(2) \leftarrow c_1(2) - 66 + 55 = 19$, *et* $\delta_{12} \leftarrow 66$
$c_1(3) \leftarrow c_1(3) - 76 + 75 = 0$, *et* $\delta_{13} \leftarrow 76$
$c_2(2) \leftarrow c_2(2) - 71 + 95 = 24$, *et* $\delta_{22} \leftarrow 71$
$c_\varnothing \leftarrow c_\varnothing + 1 = 123$
$\delta_\varnothing \leftarrow \delta_\varnothing + 1 = 123$

4 Résultats expérimentaux

Nous avons implémenté cette approche dans TOULBAR2, un solveur exact de WCSP en C++ [3]. Pour les tests suivants nous avons imposé un temps limite de 30 minutes sur un seul coeur d'un Intel Xeon E5-2680 v3 à 2.50 GHz et 256 GB de RAM.

4.1 Compétition pseudo-Booléenne 2016

Nous avons testé TOULBAR2 sur les 1600 instances OPT-SMALLINT-LIN proposées lors de la compétition pseudo-Booléenne 2016 [4]. Toutes les instances sont booléennes et composées exclusivement de contraintes linéaires, le nombre de variables, de contraintes et l'arité des contraintes varient grandement. Nous comparons TOULBAR2 v1.1.2 aux solveurs PB NAPS v1.02b (vainqueur de la compétition en 2016), ROUNDINGSAT v2 et aux solveurs ILP, CPLEX v12.7.0.0 et SCIP v7.0.2. Nous avons appliqué les paramètres pour chercher une solution exacte avec CPLEX ($epagap = epgap = epint = 0$ et $eprhs = 10^{-9}$), les paramètres par défaut ont été utilisés pour les autres solveurs. Nous avons transformé les instances de la compétition PB en WCSP, en interprétant la fonction objective sous la forme de coûts unaires et en transformant les contraintes linéaires en contraintes de la forme $\sum_{x_i \in S} w_i x_i \geq C$. Sur toutes les instances, 1200 sont résolues par au moins un solveur. La table 1 récapitule pour chaque solveur le nombre totale d'instances résolues, le nombre de fois où il est le plus rapide [5], le nombre de fois qu'il est le seul à résoudre un problème et donne une comparaison deux à deux du temps de résolution. Notre approche ne semble pas idéale pour ce genre de problème, l'efficacité de TOULBAR2 dépend largement de la famille considérée, il est parfois compétitif (*caixa,primesdimacsnf...*) ou bien peu adapté (*rand,radar...*) avec parfois 6 fois moins d'instances résolues que les autres solveurs. D'après la table 1, TOULBAR2 est presque dominé par CPLEX et ROUNDINGSAT, cela peut être dû au fait que la compétition PB16 est principalement composée d'instances avec un grand nombre de contraintes de portée très grande. TOULBAR2 propage les contraintes une par une, le minorant ainsi obtenu est sous-optimal comparé à celui calculé en résolvant le LP comme le fait CPLEX. Et malgré une complexité de propagation raisonnable, ce rapport temps de calcul/qualité du minorant ne semble pas adapté ici. Les 4 instances uniquement résolues par TOULBAR2 appartiennent à la sous-famille *auto-corr_bern* (31 instances) de la famille *minlplib2-pb-0.1.0*, où les contraintes sont de faibles arités (max

TABLE 1 – Comparaison des temps de résolution sur la compétition PB OPT-SMALLINT-LIN 2016, données à lire en colonne.

	ROUNDINGSAT	NAPS	TB2	SCIP	CPLEX
ROUNDINGSAT	–	249	64	322	483
NAPS	618	–	205	537	677
TB2	850	643	–	746	881
SCIP	656	490	219	–	788
CPLEX	413	316	103	129	–
Instances résolues	1030	890	724	1057	1100
Le plus rapide	235	178	26	25	378
Seul à résoudre	17	21	4	2	31

5). Des tests similaires ont été effectué sur la famille lion9-single-obj composée de 126 instances de la compétition OPT-BIGINT-LIN, ces résultats montrent que notre approche peut aussi s'appliquer à des coefficients de grande taille sans pour autant dépasser les solveurs de l'état de l'art sauf à de rares exceptions.

4.2 Problème de placement des entrepôts à capacité limité

Dans le problème de placement des entrepôts (Warehouse Location Problem (WLP)), une entreprise veut ouvrir des entrepôts pour approvisionner ses magasins. L'objectif est de déterminer à partir d'une liste de potentiels entrepôts lesquels il faut ouvrir afin de minimiser le coût d'entretien et d'approvisionnement des différents magasins. Chaque entrepôt à une capacité qui lui est propre, de ce fait la demande des magasins qu'il approvisionne ne doit pas dépasser sa capacité. Il est possible de formaliser cette contrainte par une contrainte de sac-à-dos à choix multiples. Nous ajoutons également afin de faciliter la recherche une contrainte linéaire (redondante) imposant que la somme des capacités des entrepôts ouverts soit supérieure à la demande totale des magasins. Nous avons testé notre approche sur 15 instances [24] ayant 100 (5 *capmo* instances), 200(*capmp*), 300 (*capmq*) entrepôts et jusqu'à 90,901 fonctions de coûts parmi lesquels on trouve 301 contraintes linéaires d'arité 300. Nous utilisons la modélisation CFN proposée par [12] à laquelle nous ajoutons les contraintes PB. Les solveurs PB et ILP utilisent une formulation directe avec des contraintes linéaires. La table 2 récapitule le nombre d'instances résolues en moins de 10 heures. Nous donnons le temps moyen et le nombre de backtracks moyen (pour les instances résolues). Nous comparons notre approche avec NAPS v1.02b , ROUNDINGSAT v2 , CPLEX v12.7.0.0, SCIP v7.0.2 et le solveur CP OR-TOOLS v9.0.9048 (utilisant l'interface flatzinc). CPLEX obtient les meilleurs résultats, étant 19 (resp. 115) plus rapide que SCIP (resp. OR-TOOLS). TOULBAR2 ne résout pas 3/15 instances dans la limite de temps. Bien qu'il parcourt un

3. https ://github.com/toulbar2/toulbar2
4. http ://www.cril.univ-artois.fr/PB16/
5. Instance non prise en compte si au moins 2 solveurs l'ont résolue en moins de 0.005 secondes (357 cas).

TABLE 2 – Nombre d'instances résolues, temps CPU moyen, et nombre de retours-arrières moyen (ou noeuds de l'arbre de recherche pour les solveurs ILP) pour 15 problèmes de placement d'entrepôts à capacité limité.

NAPS	ROUNDINGSAT	TB2	OR-TOOLS	SCIP	CPLEX12.10
0	0	12	15	15	15
-	-	4,598 sec.	7,075 sec.	1,177 sec.	61.2 sec.
-	-	466,482 bt.	304,410 bt.	1,026 nd.	1,183 nd.

TABLE 3 – Nombre d'instances résolues, temps CPU moyen et nombre de retours-arrières moyen (ou noeuds de l'arbre de recherche pour les solveurs ILP) pour 50 problèmes de sac-à-dos.

NAPS	CHUFFED[27]	OR-TOOLS	TB2	ROUNDINGSAT	CPLEX	SCIP
0	50	50	50	50	50	50
-	180.9 sec.	0.134 sec.	0.086 sec.	0.035 sec.	0.011 sec.	0.01 sec.
-	990,631 bt.	24.2 bt.	204.7 bt.	215 bt.	16.2 nd.	1 nd.

plus grand nombre de noeuds ($\approx \times 1,000$), il résout plus rapidement que SCIP (le 2ème meilleur solveur) 3 instances (*capmp2, capmq2, capmq5*). Il est aussi plus rapide que OR-TOOLS sur 7 instances, dont 4/5 des plus grandes *capmq* (excepté *capmq1* qui n'est pas résolu). ROUNDINGSAT et NAPS n'ont résolu aucune instance en moins de 10 heures.

4.3 Problème de sac à dos

Pour l'état de l'art nous avons testé TOULBAR2 sur des problèmes de sac à dos ayant de 100 à 300 variables cités dans [9, 27]. La table 3 donne le nombre d'instances résolues et le temps moyen pour différents solveurs. TOULBAR2 résout les 50 instances assez efficacement, il est plus rapide que OR-TOOLS et CHUFFED (résultats pris dans [27] avec le modèle *length-3 nogood detection*). NaPS n'est pas adapté et ne résout aucune instance en moins de 1800s, tandis que RoundingSat est plus efficace que TOULBAR2. Sans surprise les solveurs ILP CPLEX et SCIP résolvent toutes les instances presque instantanément.

4.4 Problème de sac à dos avec graphe de conflits

Nous comparons ici les différents solveurs face aux problèmes Knapsack with Conflict Graph (KPCG) [6, 10]. Ces instances sont similaires à un problème de sac-à-dos mais associées à un grand nombre de contraintes binaires dures indiquant un conflit entre deux variables. Le nombre de variables Booléennes varie entre 120 et 1000, le poids associé à chaque variable est uniformément distribué entre $[250, 500]$ et la capacité est 1000. On définit 6 classes différentes $C1, C3, C10, R1, R3, R10$, les classes C signifient que le poids et le profit de chaque variable sont corrélés sinon le profit est aléatoire entre $[1, 1000]$ (instance R). Les nombres $1, 3, 10$ correspondent à un coefficient multiplicateur de la capacité, plus le multiplicateur est grand, plus l'instance est compliquée à résoudre. En plus de la variation du nombre de variable, la densité du graphe varie de $0,1$ à $0,9$, totalisant 720 instances par classe. Nous avons utilisé un encodage direct pour TOULBAR2 et un encodage de tuple [19] pour les autres solveurs afin de proposer une relaxation linéaire plus forte et favoriser la puissance de la propagation unitaire ainsi que des mécanismes d'apprentissage. La table 4 récapitule le nombre d'instances résolues par chaque solveur, encore une fois NAPS n'est pas adapté à ce genre de problème et ne résout que 350 instances de la classe $C1$ (une des plus faciles), de ce fait nous n'avons pas continué les expérimentations pour les autres classes. L'autre solveur PB ROUNDINGSAT n'est pas très efficace non-plus, il est dernier en terme de nombre d'instances résolus. Les solveurs ILP sont plus efficaces que les solveurs PB mais bien qu'offrant une relaxation linéaire plus forte, l'encodage en tuple ne s'avère pas payant ici. En effet l'approche de TOULBAR2 est plus efficace que SCIP et CPLEX utilisant l'encodage de tuple et compétitive avec CPLEX (version 12.6) utilisant un encodage direct (résultats repris de [6]) pour trois classes sur six. Cela peut s'expliquer par le nombre important de variables (trois par contrainte binaire) ajouté par l'encodage en tuple.

4.5 Séquence de solutions diverses pour CPD

Une protéine est une chaîne de molécules simples appelés acides aminés, cette séquence détermine en quelle forme 3D la protéine va se replier. Le problème de conception de protéine (Computational Protein Design (CPD) [4]) consiste à identifier une chaîne d'acides aminés à partir d'une forme 3D. Ce problème peut être modélisé par un CFN avec des coûts unaires et des fonctions de coûts binaires représentant l'énergie de la protéine. Cependant le critère ne fait qu'approcher la réalité, ainsi pour augmenter les chances de trouver la bonne chaîne d'acides aminés il est préférable de produire une séquence de solutions diverses. Chaque fois qu'une solution est trouvée, une contrainte de distance de Hamming est ajoutée au modèle pour imposer que la prochaine solution soit différente de la précédente. La distance de Hamming peut directement être encodée

TABLE 4 – Nombre d'instances KPCG résolues.

	RSat_t	TB2	SCIP_t	CPLEX_t	CPLEX[6]	NAPS
C1	621	708	663	694	**720**	350
C3	360	571	556	477	**622**	-
C10	255	**485**	396	316	443	-
R1	700	**720**	619	703	**720**	-
R3	559	**709**	628	566	695	-
R10	351	524	456	368	**538**	-

par une contrainte PB avec des partitions EO, cela à été implémenté dans TOULBAR2 et comparé à des encodages basés sur les automates (encodage ternaire, hidden, et dual) [33, 34] sur 30 instances [6]. Pour toutes les instances le temps limite était 1 heure et le solveur s'arrête après avoir trouvé 10 solutions diverses, la distance de Hamming est 10 et nous utilisons VAC en preprocessing sur les contraintes binaires et unaires (options -A -d : -a=10 -div=10 -divm=(0 pour dual, 1 pour hidden, 2 pour ternaire, et 3 pour encodage PB)). Le graph 1 récapitule le temps de résolution pour chaque encodage. L'encodage dual et ternaire ont échoué à donner 10 solutions pour 1 instance en moins de 1 heure, tandis que l'encodage PB et hidden ont réussi. De plus l'encodage PB est plus rapide sur 27 instances et en résout 24 en moins de 30 secondes tandis que l'encodage dual, hidden et ternaire en résolvent respectivement 12, 13 et 4 en moins de 30 secondes. Les encodages basés sur les automates ont le défaut d'introduire des variables supplémentaires qui peuvent perturber l'heuristique de choix de variable (ici *min domain size per weighted degree* [8]) et les algorithmes de cohérences locales (ici, EDAC [12] pendant la recherche). Tandis que l'encodage PB encode directement la distance de Hamming mais ralentit la propagation comme on peut l'observer en comparant le nombre de backtracks par seconde (1,094 pour l'encodage PB et 2,516 pour l'encodage dual).

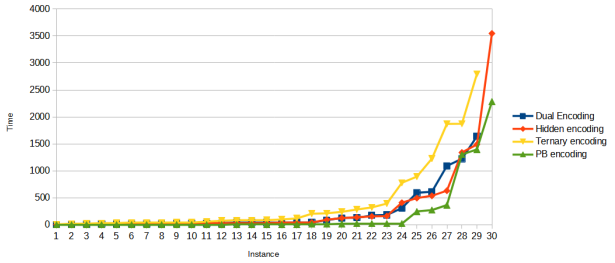

FIGURE 1 – Temps CPU de résolution pour différents encodages de la contrainte de distance de Hamming sur les 30 instances CPD.

5 Conclusion et travaux futurs

Il est à présent possible de modéliser une contrainte linéaire pseudo-Booléenne dans un WCSP. Cela offre une plus grande souplesse de modélisation et permet la résolution d'un plus grand nombre de problèmes par un solveur WCSP comme TOULBAR2. Les instances pseudo-Booléennes composées d'un faible nombre de contraintes linéaires de grande arité associées à des contraintes binaires est un cadre avantageux pour notre approche, sur ce type d'instance TOULBAR2 surpasse parfois les solveurs pseudo-Booléens et ILP. Cependant dans le cas général, ce n'est pas suffisant pour être compétitif face à ces mêmes solveurs. Une des faiblesses de notre approche est que l'algorithme propage une à une les contraintes pseudo-Booléennes sans prendre en compte les autres contraintes, produisant ainsi une solution sous-optimale du programme linéaire. Il y a plusieurs améliorations imaginables, en utilisant par exemple une relaxation Lagrangienne [23] ou une approche plus proche de VAC [15]. Cela ouvre aussi la possibilité d'utiliser les contraintes linéaires pour générer des coupes (ou autres mécanismes d'apprentissages).

6. 5 instances (*1ENH.matrix.36p.17aa, 1STN.matrix.120p.18aa, HHR.matrix.115p.19aa, 1PGB.matrix.31p.17aa, 2CI2.matrix.51p.18aa*) ont été enlevé car TOULBAR2 ne les résout pas en moins de 9,000 seconds même sans contrainte de diversité [1].

Financement

Ce travail a bénéficié d'aides de l'État gérées par l'Agence Nationale de la Recherche au titre du programme d'Investissements d'Avenir portant les références ANR-18-EURE-0021 et ANR-19-P3IA-0004.

Références

[1] David ALLOUCHE, Sophie BARBE, Simon de GIVRY, George KATSIRELOS, Yahia LEBBAH, Samir LOUDNI, Abdelkader OUALI, Thomas SCHIEX, David SIMONCINI et Matthias ZYTNICKI : *Operations Research and Simulation in Healthcare*, chapitre Cost Function Networks to Solve Large Computational Protein Design Problems. Springer, 2021.

[2] David ALLOUCHE, Christian BESSIÈRE, Patrice BOIZUMAULT, Simon de GIVRY, Patricia GUTIERREZ, Jimmy H.M. LEE, Ka Lun LEUNG, Samir LOUDNI, Jean-Philippe MÉTIVIER, Thomas SCHIEX et Yi WU : Tractability-preserving transformations of global cost functions. *Artificial Intelligence*, 238:166–189, 2016.

[3] David ALLOUCHE, Christian BESSIÈRE, Patrice BOIZUMAULT, Simon DE GIVRY, Patricia GUTIERREZ, Samir LOUDNI, Jean-Philippe METIVIER et Thomas SCHIEX : Filtering decomposable global cost functions. *In Proc. of AAAI-12*, Toronto, Canada, 2012.

[4] David ALLOUCHE, Jessica DAVIES, Simon de GIVRY, George KATSIRELOS, Thomas SCHIEX, Seydou TRAORÉ, Isabelle ANDRÉ, Sophie BARBE, Steve PRESTWICH et Barry O'SULLIVAN : Compu-

tational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79, 2014.

[5] Carlos Ansótegui, Miquel Bofill, Jordi Coll, Nguyen Dang, Juan Luis Esteban, Ian Miguel, Peter Nightingale, András Z. Salamon, Josep Suy et Mateu Villaret : Automatic detection of at-most-one and exactly-one relations for improved SAT encodings of pseudo-boolean constraints. *In* Thomas Schiex et Simon de Givry, éditeurs : *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, volume 11802 de *Lecture Notes in Computer Science*, pages 20–36. Springer, 2019.

[6] Andrea Bettinelli, Valentina Cacchiani et Enrico Malaguti : A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing*, 29(3):457–473, 2017.

[7] Miquel Bofill, Jordi Coll, Josep Suy et Mateu Villaret : An mdd-based SAT encoding for pseudo-boolean constraints with at-most-one relations. *Artif. Intell. Rev.*, 53(7):5157–5188, 2020.

[8] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre et Lakhdar Sais : Boosting systematic search by weighting constraints. *In ECAI*, volume 16, page 146, 2004.

[9] Geoffrey Chu et Peter J Stuckey : Dominance driven search. *In International Conference on Principles and Practice of Constraint Programming*, pages 217–229. Springer, 2013.

[10] Stefano Coniglio, Fabio Furini et Pablo San Segundo : A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research*, 289(2):435–455, 2021.

[11] Martin C Cooper, Simon De Givry, Martı Sánchez, Thomas Schiex, Matthias Zytnicki et Tomas Werner : Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478, 2010.

[12] Simon De Givry, Federico Heras, Matthias Zytnicki et Javier Larrosa : Existential arc consistency : Getting closer to full arc consistency in weighted csps. *In IJCAI*, volume 5, pages 84–89, 2005.

[13] Simon de Givry et George Katsirelos : Clique cuts in weighted constraint satisfaction. *In International Conference on Principles and Practice of Constraint Programming*, pages 97–113. Springer, 2017.

[14] Heidi Dixon : *Automating pseudo-boolean inference within a dpll framework*. Citeseer, 2004.

[15] Tomás Dlask et Tomás Werner : Bounding linear programs by constraint propagation : Application to max-sat. *In* Helmut Simonis, éditeur : *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 de *Lecture Notes in Computer Science*, pages 177–193. Springer, 2020.

[16] Tomás Dlask et Tomás Werner : On relation between constraint propagation and block-coordinate descent in linear programs. *In* Helmut Simonis, éditeur : *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 de *Lecture Notes in Computer Science*, pages 194–210. Springer, 2020.

[17] Niklas Eén et Niklas Sörensson : Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[18] Jan Elffers et Jakob Nordström : Divide and conquer : Towards faster pseudo-boolean solving. *In IJCAI*, volume 18, pages 1291–1299, 2018.

[19] Barry Hurley, Barry O'Sullivan, David Allouche, George Katsirelos, Thomas Schiex, Matthias Zytnicki et Simon De Givry : Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints*, 21(3):413–434, 2016.

[20] Ambros Gleixner Jo Devriendt et Jakob Nordström : Learn to relax : Integrating 0-1 integer linear programming with pseudo-boolean conflict-driven search. *In Proceeding of CPAIOR 2020*, 2020.

[21] Ellis L Johnson et Manfred W Padberg : A note of the knapsack problem with special ordered sets. *Operations Research Letters*, 1(1):18–22, 1981.

[22] V Kolmogorov : Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1568–1583, 2006.

[23] Nikos Komodakis, Nikos Paragios et Georgios Tziritas : Mrf energy minimization and beyond via dual decomposition. *IEEE transactions on pattern analysis and machine intelligence*, 33(3): 531–552, 2010.

[24] J. Kratica, D. Tosic, V. Filipovic et I. Ljubic : Solving the Simple Plant Location Problems by Genetic Algorithm. *RAIRO Operations Research*, 35:127–142, 2001.

[25] J. Larrosa : On arc and node consistency in weighted CSP. *In Proc. AAAI'02*, pages 48–53, Edmondton, (CA), 2002.

[26] JHM Lee et Yu Wai Shum : Modeling soft global constraints as linear programs in weighted constraint satisfaction. *In 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pages 305–312. IEEE, 2011.

[27] Jimmy HM Lee et Allen Z Zhong : Automatic dominance breaking for a class of constraint optimization problems. *In Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 1192–1200, 2020.

[28] Jimmy Ho-Man Lee et Ka Lun Leung : Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *Journal of Artificial Intelligence Research*, 43:257–292, 2012.

[29] Joao P Marques-Silva et Karem A Sakallah : Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[30] David Pisinger et Paolo Toth : Knapsack problems. *In Handbook of combinatorial optimization*, pages 299–428. Springer, 1998.

[31] Daniel Prusa et Tomas Werner : Universality of the local marginal polytope. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1738–1743, 2013.

[32] Francesca Rossi, Peter Van Beek et Toby Walsh : *Handbook of constraint programming*. Elsevier, 2006.

[33] Manon Ruffini, Jelena Vucinic, Simon de Givry, George Katsirelos, Sophie Barbe et Thomas Schiex : Guaranteed diversity & quality for the weighted csp. *In 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 18–25. IEEE, 2019.

[34] Manon Ruffini, Jelena Vucinic, Simon de Givry, George Katsirelos, Sophie Barbe et Thomas Schiex : Guaranteed diversity and optimality in cost function network based computational protein design methods. *Algorithms*, 2021.

[35] Masahiko Sakai et Hidetomo Nabeshima : Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE TRANSACTIONS on Information and Systems*, 98(6):1121–1127, 2015.

[36] M. Sánchez, S. de Givry et T. Schiex : Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1):130–154, 2008.

[37] D Sontag, D Choe et Y Li : Efficiently searching for frustrated cycles in MAP inference. *In Proc. of UAI*, pages 795–804, 2012.

[38] D Sontag, T Meltzer, A Globerson, Y Weiss et T Jaakkola : Tightening LP relaxations for MAP using message-passing. *In Proc. of UAI*, pages 503–510, Helsinki, Finland, 2008.

[39] Fulya Trösser, Simon de Givry et George Katsirelos : Relaxation-aware heuristics for exact optimization in graphical models. *In* Emmanuel Hebrard et Nysret Musliu, éditeurs : *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21-24, 2020, Proceedings*, volume 12296 de *Lecture Notes in Computer Science*, pages 475–491. Springer, 2020.

[40] Tomas Werner : A Linear Programming Approach to Max-sum Problem : A Review. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 29(7):1165–1179, juillet 2007.

[41] M. Zytnicki, C. Gaspin, S. de Givry et T. Schiex : Bounds Arc Consistency for Weighted CSPs. *Journal of Artificial Intelligence Research*, 35:593–621, 2009.