Filtrage de fonctions de coût globales décomposables

David Allouche, Christian Bessiere, Patrice Boizumault, Simon de Givry, Patricia Gutierrez, Samir Loudni, Jean-Philippe Metivier, Thomas Schiex

Filtrage de fonctions de coût globales décomposables

D. Allouche[1] C. Bessiere[2] P. Boizumault[3] S. de Givry[1]
P. Gutierrez[4] S. Loudni[3] JP. Métivier[3] T. Schiex[1]

[1]UBIA UR 875, INRA, F-31320 Castanet Tolosan, France
[2]U. Montpellier, France
[3]GREYC-CNRS, UMR 6072, U. Caen, France
[4]IIIA-CSIC, U. Autònoma de Barcelona, Bellaterra, Spain

Abstract

As [18] have shown, weighted constraint satisfaction problems can benefit from the introduction of global cost functions, leading to a new Cost Function Programming paradigm. In this paper, we explore the possibility of decomposing global cost functions in such a way that enforcing soft local consistencies on the decomposition achieves the same level of consistency on the original global cost function. We give conditions under which directional arc consistency and virtual arc consistency offer such guarantees. We conclude by experiments on decomposable cost functions showing that decompositions may be very useful to easily integrate efficient global cost functions in solvers.

Résumé

Les auteurs de [18] ont montré que les problèmes de satisfaction de contraintes pondérées peuvent bénéficier de l'introduction des fonctions de coût globales, conduisant au nouveau paradigme de la programmation par fonctions de coûts. Dans cet article, nous explorons la possibilité de décomposer les fonctions de coût globales de sorte qu'appliquer une consistance locale souple sur la décomposition produit le même niveau de consistance que sur la fonction de coût initiale. Nous donnons des conditions pour lesquelles l'arc consistance directionelle et l'arc consistance virtuelle offrent de telles garanties. Nos expérimentations, menées sur des fonctions de coût décomposables, montrent que les décompositions peuvent être très utiles pour intégrer efficacement des fonctions de coût globales dans des solvers.

†. Ce travail a été soutenu par l'Agence nationale de la Recherche, reference ANR-10-BLA-0214.

1 Introduction

Le traitement de modèles graphiques est un problème central en intelligence artificielle. L'optimisation d'une combinaison de fonctions de coût locales, très étudiée dans les réseaux de contraintes pondérées [25], permet de capturer une grande variété de problèmes, tels que SAT, Max-SAT, CSP pondérés, recherche d'une explication de probabilité maximale (MPE pour *Maximum Probability Explanation*) dans les réseaux Bayésiens, le calcul d'un maximum *a posteriori* (MAP pour *Maximum A posteriori Problem*) dans les champs markoviens [14] ou encore l'optimisation de fonctions pseudo-booléennes [6]. Ses applications sont nombreuses en allocation des ressources et en bioinformatique.

La principale approche utilisée pour résoudre ces problèmes s'appuie sur les méthodes fondées sur le principe de séparation et évaluation, combiné avec des techniques dédiées de calcul de minorants permettant d'élaguer l'arbre de recherche. Ces minorants peuvent être fournis par l'application de cohérences locales souples [7], comme dans les solveurs de Programmation par Contraintes (CP). De tels solveurs disposent de contraintes globales qui représentent un outil indispensable pour la modélisation et la résolution de problèmes difficiles de grande taille. Des algorithmes dédiés pour le filtrage de ces contraintes ont été proposés. Pour certaines contraintes globales telles que, REGULAR, CONTIGUITY, AMONG, il a été démontré qu'une décomposition en un réseau Berge-acyclique de con-

traintes d'arité fixe peut conduire à une mise en œuvre simple, sans aucune perte d'efficacité du filtrage [2, 4].

La notion de contrainte globale a été récemment étendue aux CSP pondérés, définissant les fonctions de coût globales [29, 28, 20, 19] avec des algorithmes de filtrage efficaces. Dans cet article, après un rappel de quelques notions préliminaires, nous introduisons la notion de décomposition de fonctions de coût globales et nous montrons comment les contraintes globales décomposables peuvent être relâchées sous la forme de fonctions de coût globales décomposables, ayant la même structure de décomposition. Pour les fonctions de coût globales décomposables Berge-acycliques, nous montrons que l'application de la cohérence d'arc directionnelle ou de la cohérence d'arc virtuelle sur la décomposition produit le même niveau de consistance que sur la fonction de coût de départ. Enfin, nous comparons expérimentalement l'efficacité des versions décomposées et monolithiques de plusieurs fonctions de coût globales et les résultats observés montrent des accélérations importantes en faveur de la décomposition.

2 Préliminaires

2.1 Réseau de fonctions de coût

Un réseau de fonctions de coût (CFN) est une paire (X, W) où $X=\{1, \ldots, n\}$ est un ensemble de n variables et W est un ensemble de fonctions de coût. Chaque variable $i \in X$ a un domaine fini $D_i \in D$ de valeurs qui peuvent lui être affectées. La taille du plus grand domaine est notée d. Une affectation de i à la valeur $a \in D_i$ est notée (i, a). Pour un sous-ensemble de variables $S \subseteq X$, on note D^S le produit cartésien des domaines des variables de S. Pour un n-uplet donné t, $t[S]$ représente la projection habituelle du n-uplet t sur l'ensemble de variables S.

Une fonction de coût $w_S \in W$, de portée $S \subseteq X$, est une fonction $w_S : D^S \mapsto [0, k]$ où k est un coût entier maximum (fini ou non) utilisé pour représenter les affectations interdites (exprimant des contraintes dures). Pour capturer fidèlement les contraintes dures, les coûts sont combinés par l'addition bornée $\oplus$, définie par $\alpha \oplus \beta = \max(k, \alpha + \beta)$. Dans cet article, une *contrainte* dure est donc représentée par une fonction de coût prenant des valeurs dans l'ensemble $\{0, k\}$ seulement. Si $\forall t \in D^S, z_S(t) \leq w_S(t)$, la fonction de coût z_S est appelée une relaxation de w_S, notée $z_S \leq w_S$. Un coût β peut être soustrait d'un coût α plus grand en utilisant l'opération $\ominus$ où $\alpha \ominus \beta$ est égale à $(\alpha - \beta)$ si $\alpha \neq k$. Sinon il est égal à k. Nous supposons qu'il existe, pour chaque variable i, une fonction de coût unaire w_i ainsi qu'une fonction de coût d'arité nulle notée $w_\varnothing$.

Le problème consiste à trouver une affectation complète t de l'ensemble des variables minimisant la combinaison des fonctions de coût $\bigoplus_{w_S \in W} w_S(t[S])$. Le problème de décision, associé à ce problème d'optimisation, est NP-complet. Sa restriction aux variables booléennes et aux contraintes binaires est connue pour être APX-difficiles [21].

Un réseau de contraintes (CN) est un CFN où toutes les fonctions de coût représentent des contraintes dures. Ces fonctions de coût seront appelées contraintes.

2.2 Consistance locale

Les algorithmes de recherche de solutions dans les réseaux de contraintes (CN) utilisent les techniques de renforcement de consistance locale afin de réduire l'espace de recherche. Dans ces réseaux, la cohérence d'arc généralisée (GAC) est le niveau de consistance locale le plus utilisé. Une contrainte c_S est dite GAC, ssi, chaque valeur du domaine de chaque variable de S possède au moins un support dans c_S. Un support dans c_S est un n-uplet $t \in D^S$ tel que $c_S(t) = 0$.

Établir la cohérence d'arc généralisée sur c_S sera désigné par le terme *filtrer* c_S. Habituellement, la résolution de problèmes de minimisation dans les réseaux de fonctions de coût repose sur des méthodes exactes fondées sur le principe de séparation et évaluation combiné avec le calcul de minorants obtenus par renforcement de consistances locales telles que la cohérence d'arc directionnelle (DAC) [8, 17] et la cohérence d'arc virtuelle (VAC) [7].

2.3 Fonction de coût globale

Une *contrainte* globale $c(S, \theta)$ représente une famille de contraintes ayant une sémantique donnée, définie sur un ensemble de variables S et incluant d'éventuels paramètres additionnels représentés par θ. Les contraintes globales utilisent des algorithmes de filtrage dédiés plus efficaces que les algorithmes de filtrage génériques. Plusieurs contraintes globales ont été étendues pour capturer une mesure de violation comme par exemple SoftAllDiff(S) [23] ou SoftRegular($S, \mathcal{A}, d$) [27]). Ces contraintes globales relâchable sont des *contraintes dures*, incluant une variable de coût permettant de quantifier la violation selon une sémantique de violation donnée. Pour plusieurs de ces contraintes, des algorithmes efficaces dédiés établissant la GAC ont été proposés.

Récemment, différents travaux [28, 18] ont montré qu'il est possible de capturer ces même mesures de violation au travers de fonctions de coût paramétrées $z(S, \theta)$, calculant directement le coût d'une affectation.

Cette approche permet d'exploiter, de manière immédiate, les consistances locales souples grâce à des algorithmes de filtrage dédiés fournissant des minorants de bien meilleurs qualité.

En effet, les fonctions de coût, combinées avec les consistances locales souples, offrent un filtrage plus performant que l'approche précédente basée sur les variables de coût et les contraintes globales souples. Ces améliorations sont dues à une meilleure communication entre les fonctions de coût permise par l'application de transformations préservant l'équivalence (EPT) [9, 18].

2.4 Hypergraphe

L'hyper-graphe d'un réseau de fonctions de coût (ou d'un réseau de contraintes) (X, W) est constitué d'un sommet par variable $i \in X$ et d'une hyper-arête par portée S telle que $\exists w_S \in W$. Nous ne considérons que des CFN constitués d'hyper-graphes connectés. Le graphe d'incidence d'un hyper-graphe (X, E) est un graphe $G = (X \cup E, E_H)$ où $(x_i, e_j) \in E_H$ ssi $x_i \in X, e_j \in E$ et x_i appartient à hyper-arête e_j. Un hyper-graphe (X, E) est Berge acyclique ssi son graphe d'incidence est acyclique.

3 Décomposition de fonctions de coût globales

Certaines contraintes globales peuvent être efficacement décomposées en un sous-réseau équivalent de contraintes d'arité bornée [5, 3]. De même, les fonctions de coût globales peuvent être décomposées en un ensemble de fonctions coûts d'arité bornée. À noter que la définition ci-dessous s'applique à toute fonction de coût, y compris aux contraintes (fonctions de coût utilisant uniquement les coûts $\{0, k\}$).

Définition 1 *Une décomposition d'une fonction de coût $z(T, \theta)$ est une transformation polynomiale δ_p (p étant un entier) qui retourne un CFN $\delta_p(T, \theta) = (T \cup E, F)$ tel que $\forall w_S \in F, |S| \leq p$ et $\forall t \in D^T, z(T, \theta)(t) = \min_{t' \in D^{T \cup E}, t'[T]=t} \bigoplus_{w_S \in F} w_S(t'[S])$.*

Nous supposons, sans aucune perte de généralité, que chaque variable supplémentaire $i \in E$ figure dans au moins deux fonctions de coût dans la décomposition. [1] Clairement, si $z(T, \theta)$ apparaît dans un CFN $P = (X, W)$ et se décompose en $(T \cup E, F)$, alors les solutions optimales de P peuvent être directement obtenues par projection des solutions optimales du CFN $P' = (X \cup E, W \setminus \{z(T, \theta)\} \cup F)$ sur X.

Exemple Considérons la contrainte AllDiff(S) ainsi que sa version souple SoftAllDiff(S, dec) avec comme mesure de violation la sémantique de *décomposition* [23] où le coût d'une affectation est le nombre de couples de variables ayant la même valeur. Il est bien connu que AllDiff se décompose en un ensemble de $\frac{n.(n-1)}{2}$ contraintes binaires de différence. De manière analogue, la fonction de coût globale SoftAllDiff(S, dec) peut être décomposée en un ensemble de $\frac{n.(n-1)}{2}$ fonctions de coût de différence. Une fonction de coût de différence prend la valeur 1, ssi, les deux variables impliquées ont la même valeur et 0 sinon. Cette décomposition ne nécessite aucune variable supplémentaire. Enfin, notons que les deux décompositions ont la même structure d'hyper-graphe.

3.1 Relaxation de contraintes globales décomposables

Nous allons à présent montrer qu'il existe une façon systématique d'obtenir des fonctions de coût décomposables, vues comme des relaxations spécifiques de contraintes globales décomposables existantes.

Comme l'a montré le précédent exemple avec AllDiff, si l'on considère une contrainte globale décomposable, alors il est possible de définir une fonction de coût globale décomposable, en relaxant chacune des contraintes issues de la décomposition.

Théoreme 1 *Soit $c(T, \theta)$ une contrainte globale qui se décompose en un réseau de contraintes $(T \cup E, C)$ et f_θ une fonction qui fait correspondre à chaque $c_S \in C$ une fonction de coût w_S telle que $w_S \leq c_S$. Alors, la fonction de coût globale $w(T, f_\theta)(t) = \min_{t' \in D^{T \cup E}, t'[T]=t} \bigoplus_{c_S \in C} f_\theta(c_S)(t'[S])$ est une relaxation de $c(T, \theta)$.*

Preuve Pour chaque n-uplet $t \in D^T$, si $c(T, \theta)(t)$=0, alors $\min_{t' \in D^{T \cup E}, t'[T]=t} \bigoplus_{c_S \in C} c_S(t'[S]) = 0$ car $(T \cup E, C)$ est une décomposition de $c(T, \theta)$. Soit $t' \in D^{T \cup E}$ le n-uplet pour lequel le minimum est atteint. Ceci entraîne que $\forall c_S \in C, c_S(t'[S]) = 0$. Comme $f_\theta(c_S)$ est une relaxation de c_S, ceci entraîne que $f_\theta(c_S)(t'[S])$=0. Ainsi, $\bigoplus_{c_S \in C} f_\theta(c_S)(t'[S]) = 0$ et $w(T, f_\theta)(t) = 0$. □

Par définition, la fonction de coût globale $w(T, f_\theta)$ est décomposable en $(T \cup E, W)$ où W est obtenu par l'application de de f_θ sur chaque élément de C. Il est à noter que, puisque f_θ préserve les portées, l'hypergraphe de la décomposition est lui aussi préservé. Ce résultat permet d'obtenir immédiatement un grand nombre de décompositions pour les fonctions de coût

1. Sinon, une telle variable peut être retirée par élimination de variable : pour cela, supprimer i de E et remplacer la fonction w_S, contenant i, par une fonction de coût $\min_i w_S$ sur $S \setminus \{i\}$. Ce qui préserve la Berge-acyclicité.

globales, ceci à partir des décompositions déjà réalisées de contraintes globales telles que AllDiff, Regular, Grammar, Among, Stretch. La paramétrisation grâce à f_θ permet une grande flexibilité.

Considérons la contrainte AllDiff(S) décomposée en une clique de contraintes binaires de différence. A partir de n'importe quel graphe $G = (V, E)$, il est possible de définir une fonction de relaxation f_G qui préserve les contraintes de différence $i \neq j$ lorsque $(i, j) \in E$, et qui, sinon les relaxe en une fonction de coût constante et égale à zéro. La fonction de coût globale ainsi construite $w(V, f_G)$ permet de modéliser le problème de coloration des sommets d'un graphe qui est un problème NP-difficile. Ainsi, établir DAC ou VAC sur cette seule fonction de coût sera aussi un problème *intraitable*, alors qu'établir DAC ou VAC sur sa décomposition en fonctions de coût binaires sera évidemment un problème polynomial mais qui affaiblira le niveau de filtrage établi.

Considérons la contrainte globale Regular($\{X_1, \ldots, X_n\}, \mathcal{A}$), définie par un automate fini $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ où Q est un ensemble d'états, Σ un alphabet, $\delta :: \Sigma \times Q \rightarrow 2^Q$ une fonction de transition, q_0 l'état initial et F l'ensemble des états terminaux. Les auteurs de [4] ont montré que cette contrainte pouvait se décomposer en un réseau de contraintes $(\{X_1, \ldots, X_n\} \cup \{Q_0, \ldots, Q_n\}, C)$ où les variables additionnelles Q_i ont pour domaine Q. L'ensemble de contraintes C contient deux contraintes unaires restreignant Q_0 à $\{q_0\}$ et Q_n à F ainsi qu'une séquence de contraintes ternaires de la forme $c_{\{Q_i, X_{i+1}, Q_{i+1}\}}$ où un triplet (q, s, q') est autorisé, ssi, $q' \in \delta(q, s)$. Une relaxation de cette décomposition consiste à relâcher chacune de ces contraintes. Les contraintes unaires portant sur Q_0 et Q_n seront remplacées par les fonctions de coût unaires λ_{Q_0} et ρ_{Q_n} donnant le coût d'utilisation de chaque état comme état initial ou comme état final. Les contraintes ternaires seront relaxées sous forme de fonctions de coût ternaires $\sigma_{\{Q_i, X_{i+1}, Q_{i+1}\}}$ donnant le coût d'utilisation de n'importe quelle transition (q, s, q'). Cette relaxation correspond précisément à l'utilisation d'un automate fini pondéré $\mathcal{A} = (Q, \Sigma, \lambda, \sigma, \rho)$ [11]. Le coût d'une affectation (mot) est égal, par définition, au coût optimal de la reconnaissance de ce mot par l'automate pondéré. On obtient ainsi la fonction de coût globale WeightedRegular($\{X_1, \ldots, X_n\}, \mathcal{A}$). Comme l'ont montré les auteurs de [13], la distance de Hamming et la distance d'édition (Edit) peuvent être toutes deux capturées par un automate pondéré. Nous verrons que, contrairement à celle de AllDiff, la décomposition de WeightedRegular peut être traitée de manière efficace par les consistances locales souples.

4 Cohérence locale et décompositions

L'utilisation de décompositions au lieu de leur version monolithique offre des avantages et des inconvénients. Du fait de la localité des calculs, la version décomposée peut être filtrée plus efficacement mais cela peut aussi limiter l'effet du filtrage. Dans les CSP classiques, il est bien connu que si la décomposition a une structure Berge-acyclique, l'établissement de la cohérence d'arc généralisée (GAC) sur la décomposition établit également GAC sur la contrainte globale [1]. Nous montrons qu'un résultat similaire peut être obtenu pour les réseaux de fonctions de coût en utilisant DAC ou VAC.

DAC a été originellement introduit sur les fonctions de coût binaires en s'appuyant sur la notion de support complet [7]. Pour une fonction de coût w_S, un n-uplet $t \in D^S$ est un support complet pour une valeur (i, a) de $i \in S$ ssi $w_i(a) = w_S(t) \bigoplus_{j \in S} w_j(t[j])$. Remarquez que soit $w_i(a) = k$ et (i, a) ne participe à aucune solution ou $w_i(a) < k$ et alors $w_S(t) \bigoplus_{j \in S, j \neq i} w_j(t[j]) = 0$. DAC a été étendu aux fonctions de coût non binaires dans [24] et [20] avec des définitions différentes qui coïncident néanmoins dans le cas de fonctions de coût binaires. Dans cet article, nous nous appuyons sur une extension simple appelée T-DAC (pour Terminal DAC). Étant donné un ordre total $\prec$ sur les variables, un CFN est dit T-DAC par rapport à l'ordre $\prec$ ssi, pour toute fonction de coût w_S, toute valeur (i, a) de la variable maximum $i \in S$ selon $\prec$ a un support complet sur w_S.

VAC est une cohérence locale plus récente qui établit un lien entre un CFN $P = (X, W)$ et un réseau de contrainte dénoté $Bool(P)$ qui a le même ensemble X de variables et qui contient exactement, pour chaque fonction de coût $w_S \in W, |S| > 0$, une contrainte c_S avec la même portée qui interdit précisément tous les n-uplets $t \in D^S$ tels que $w_S(t) \neq 0$. Un CFN P est dit VAC ssi la fermeture arc cohérente du réseau de contraintes $Bool(P)$ n'est pas vide [7].

4.1 Filtrage par cohérences locales souples

L'établissement de cohérences locales souples s'appuie sur des transformations préservant l'équivalence (*Equivalence Preserving Transformations* ou EPT) qui s'appliquent à une fonction de coût [9] w_S. Au lieu d'effacer des valeurs, une EPT déplace des coûts entre w_S et la fonction de coût unaire $w_i, i \in S$. Elle opère donc sur un sous-réseau de P défini par w_S dénoté $N_P(w_S) = (S, \{w_S\} \cup \{w_i\}_{i \in S})$. La principale EPT est décrite comme l'Algorithme 1. Cette EPT déplace une quantité de coût $|\alpha|$ entre la fonction de coût unaire w_i et la fonction de coût w_S. La direction de déplacement des coûts est indiquée par le signe de α. Les précondi-

tions garantissent que les coûts restent positifs dans le réseau obtenu après application.

Algorithme 1 : Une opération de transformation (EPT) permettant de déplacer des coûts pour établir les cohérences d'arc souples. Les opérations $\oplus, \ominus$ sont étendues pour accepter des coûts négatifs comme suit : pour des coûts non négatifs α, β, on a $\alpha \ominus (-\beta) = \alpha \oplus \beta$ et pour $\beta \leq \alpha$, $\alpha \oplus (-\beta) = \alpha \ominus \beta$.

1 Précondition : $-w_i(a) \leq \alpha \leq \min_{t \in D^s, t[i]=a} w_S(t)$;
2 **Procedure** Project(w_S, i, a, α)
3 $\quad w_i(a) \leftarrow w_i(a) \oplus \alpha$;
4 $\quad$ **pour chaque** ($t \in D^S$ tel que $t[i] = a$) **faire**
5 $\quad\quad w_S(t) \leftarrow w_S(t) \ominus \alpha$;

Pour établir T-DAC sur une fonction de coût w_S, il suffit de d'abord déplacer les coûts issus de toute fonction unaire $w_i, i \in S$ dans w_S en appliquant Project($w_S, i, a, -w_i(a)$) pour toute valeur $a \in D_i$. Soit j la variable maximum dans in S selon $\prec$, il est alors possible d'appliquer Project(w_S, j, b, α) pour toute valeur (j, b) et $\alpha = \min_{t \in D^s, t[j]=b} w_S(t)$. Soit t un n-uplet où ce minimum est atteint. t est alors un support complet pour (j, b) : $w_j(b) = w_S(t) \bigoplus_{i \in S} w_i(t[i])$. Ce support peut uniquement être brisé si une fonction de coût unaire $w_i, i \in S, i \neq j$, $w_i(a)$ augmente pour une valeur (i, a).

Pour filtrer le CFN complet (X, W) via T-DAC, il suffit de trier W selon $\prec$ et d'appliquer le processus précédent, successivement sur chacune des fonctions de coût. Quand une fonction de coût w_S est traitée, toutes les fonctions de coût dont la variable maximum apparaît avant la variable maximum de S ont déjà été traitées ce qui garantit qu'aucun des support complets déjà établis ne pourra être brisé dans le futur. Le filtrage par T-DAC s'effectue donc en $O(ed^r)$ en temps, où $e = |W|$ et $r = \max_{w_S \in W} |S|$. En utilisant la structure de données Δ introduite dans [7], la complexité spatiale peut être réduite en $O(edr)$.

L'algorithme le plus efficace pour établir VAC [7] établit en fait une approximation de VAC appelée VAC_ε avec une complexité temporelle en $O(\frac{ekd^r}{\varepsilon})$ et une complexité spatiale en $O(edr)$. De façon alternative, la cohérence d'arc souple optimale (OSAC) peut être utilisée pour établir VAC en temps $O(e^{6.5} d^{(3r+3.5)} \log M)$ (où M est le plus grand coût fini dans le réseau).

4.2 Berge acyclicité et cohérence d'arc directionnelle

Dans cette section, nous montrons que le filtrage par T-DAC d'une décomposition Berge-acyclique d'une fonction de coût ou de la fonction de coût globale d'origine fournissent la même distribution de coût sur la dernière variable et donc le même minorant (fourni par l'établissement de la cohérence de nœud [16]).

Théoreme 2 *Si une fonction de coût globale $z(T, \theta)$ se décompose en un CFN de structure Berge-acyclique $N = (T \cup E, F)$ alors il existe un ordre sur $T \cup E$ tel que la fonction de coût unaire w_{i_n} sur la dernière variable i_n produite par filtrage via T-DAC du sous-réseau $(T, \{z(T, \theta)\} \cup \{w_i\}_{i \in T})$ est identique à la fonction de coût unaire w'_{i_n} produite par filtrage via T-DAC de la décomposition $N = (T \cup E, F \cup \{w_i\}_{i \in T})$.*

Preuve Considérons le réseau décomposé N et $I_N = (T \cup E \cup F, E_I)$ son graphe d'incidence. On sait que I_N est un arbre dont les sommets sont les variables et les fonctions de coût de N. Nous enracinons I_N en une variable de T. Les voisins (parents et fils, s'ils existent) d'une fonction de coût w_S sont les variables de S. Les voisins d'une variable i sont les fonctions de coût impliquant i. Considérons un ordre topologique arbitraire des sommets de I_N. Cet ordre induit un ordre sur les variables $(i_1, \ldots, i_n), i_n \in T$ qui est utilisé pour établir T-DAC sur N. Remarquez que pour toute fonction de coût $w_S \in F$, la variable parent de w_S dans I_N apparaît après toutes les autres variables de S.

Considérons une valeur (i_n, a) de la racine. Si $w_{i_n}(a) = k$, alors tout affectation complète qui étend cette valeur a un coût égal à $w_{i_n}(a)$. Sinon, $w_{i_n}(a) < k$. Soit w_S un fils quelconque de i_n et t_S un support complet de (i_n, a) sur w_S. On a $w_{i_n}(a) = w_S(t) \bigoplus_{i \in S} w_i(t[i])$ ce qui prouve que $w_S(t) = 0$ et $\forall i \in S, i \neq i_n, w_i(t[i]) = 0$. I_N étant un arbre, il est possible d'utiliser le même raisonnement récursivement sur tous les descendants de i_n jusqu'aux feuille de l'arbre. On prouve ainsi que l'affectation (i_n, a) peut être étendue en une affectation complète de coût $w_{i_n}(a)$ dans N. Dans chacun des cas, $w_{i_n}(a)$ est le coût d'une extension optimale de (i_n, a) dans N.

Supposons maintenant que l'on établisse T-DAC en utilisant l'ordre des variables précédent sur le réseau non-décomposé $(T, \{z(T, \theta)\} \cup \{w_i\}_{i \in T})$. Soit t un support complet de (i_n, a) sur $z(T, \theta)$. Par définition $w_{i_n}(a) = z(T, \theta) \bigoplus_{i \in T} w_i(t[i])$ ce qui montre que $w_{i_n}(a)$ est égal au coût d'une extension optimale de (i_n, a) sur $(T, \{z(T, \theta)\} \cup \{w_i\}_{i \in T})$. Par définition d'une décomposition, étant donné que $i_n \notin E$, ce coût est égal au coût d'une extension optimale de (i_n, a) dans N. □

T-DAC est donc assez puissante pour traiter des décompositions Berge-acyclique sans perdre en terme de puissance de filtrage , pourvu qu'un ordre idoine soit utilisé pour appliquer les EPT.

4.3 Berge acyclicité et cohérence d'arc virtuelle

La cohérence d'arc virtuelle offre un lien simple et direct entre réseaux de contraintes et réseaux de fonctions de coût qui permet d'étendre des propriétés classiques des CSP aux CFN sous des conditions simples.

Théoreme 3 *Dans un CFN, si une fonction de coût $z(T,\theta)$ se décompose en un CFN Berge-acyclique $N = (T \cup E, F)$ alors le filtrage via VAC du réseau $(T, \{z(T,\theta)\} \cup \{w_i\}_{i\in T})$ ou du réseau $(T \cup E, F \cup \{w_i\}_{i\in T})$ fournit le même minorant $w_\varnothing$.*

Preuve L'établissement de VAC sur le CFN $P = (T \cup E, F \cup \{w_i\}_{i\in T})$ ne modifie pas l'ensemble des portées et fournit un problème équivalent P' tel que $Bool(P')$ est Berge-acyclique, une situation dans laquelle la cohérence d'arc est une procédure de décision. Nous pouvons donc directement utiliser la Proposition 10.5 de [7] qui précise que si un CFN P est VAC et si $Bool(P)$ est dans une classe de CSP résolu par la cohérence d'arc, alors P a une solution optimale de coût $w_\varnothing$.

De façon similaire, le réseau $Q = (T, \{z(T,\theta)\} \cup \{w_i\}_{i\in T})$ contient une seule fonction de coût d'arité strictement supérieure à 1 et $Bool(Q)$ sera résolu par l'établissement de la cohérence d'arc. L'établissement de VAC produira donc un CFN qui a également une solution optimale de coût $w_\varnothing$. Les réseaux P et Q ont, par définition d'une décomposition, le même coût optimal. □

5 Résultats expérimentaux

Dans cette partie nous évaluons l'intérêt pratique des décompositions de fonctions de coût. Comparées aux propagateurs monolithiques des fonctions de coût globales, ces décompositions sont faciles à implanter et procurent un bon filtrage. Mais leur efficacité en temps reste à évaluer.

Tous les problèmes utilisés ici ont été résolus en utilisant le solveur de fonctions de coût `toulbar2 0.9.5`[2] avec les pré-traitements désactivé (option `-o -e: -f: -dec: -h: -c: -d: -q:`), et avec un ordre d'affectation des variables et un ordre DAC compatibles avec la structure Berge-acyclique des décompositions. L'ordonnancement dynamique des valeurs choisit les valeurs supports pour la cohérence d'arc existentielle (EAC) en premier [15]. Nous n'utilisons pas de majorant initial. Dans tous les cas nous utilisons le même niveau de cohérence locale (à savoir (weak) EDGAC*, plus fort que T-DAC et qui produit donc un $w_\varnothing$ optimal pour chaque fonction de coût globale). Toutes les expériences ont été faites en utilisant plusieurs cœurs CPU Intel Xeon 2.66 Ghz avec 64GB de RAM.

2. https ://mulcyber.toulouse.inra.fr/projects/toulbar2.

5.1 WeightedRegular aléatoire

Comme dans [22], nous générons des automates aléatoires avec $|Q|$ états et $|\Sigma|$ symboles. Nous sélectionnons aléatoirement 30% des paires possibles $(s, q_i) \in \Sigma \times Q$ et choisissons aléatoirement un état $q_j \in Q$ pour former une transition $\delta(s, q_i) = q_j$ pour chacune de ces paires. L'ensemble F des états finaux est obtenu en sélectionnant aléatoirement 50% des états de Q. La génération aléatoire suit une distribution uniforme.

A partir de chaque automate, nous construisons deux CFN : un qui utilise une fonction de coût SoftRegular monolithique avec distance de Hamming [20] et un autre qui utilise la décomposition Berge-acyclique d'une fonction de coût WeightedRegular équivalente. Pour rendre le problème moins artificiel nous ajoutons à chacun de ces problèmes le même ensemble de contraintes unaires aléatoires (une par variable d'origine, les coûts unaires étant aléatoirement choisis entre 0 et 9). Nous mesurons deux temps : (1) le temps pour charger et filtrer le problème initial, et (2) le temps total pour résoudre le CFN (incluant le temps (1)). Le temps (1) donne des informations sur la complexité du filtrage alors que le temps (2) donne des informations sur l'incrémentalité des algorithmes de filtrage. Les temps reportés sont des moyennes sur 100 instances. Les instances atteignant la limite de temps de une heure sont comptés pour 3600 secondes.

n	$\|\Sigma\|$	$\|Q\|$	Monolithique		Décomposé	
			filtrage	solve	filtrage	solve
25	5	10	0,12	0,51	0,00	0,00
		80	2,03	9,10	0,08	0,08
25	10	10	0,64	2,56	0,01	0,01
		80	10,64	43,52	0,54	0,56
25	20	10	3,60	13,06	0,03	0,03
		80	45,94	177,5	1,51	1,55
50	5	10	0,45	3,54	0,00	0,00
		80	11,85	101,2	0,17	0,17
50	10	10	3,22	20,97	0,02	0,02
		80	51,07	380,5	1,27	1,31
50	20	10	15,91	100,7	0,06	0,07
		80	186,2	1 339	3,38	3,47

Si on regarde juste le temps de filtrage, il est clair que les décompositions offrent une très forte accélération malgré une implantation bien plus simple que les propagateurs monolithiques. Le temps de résolution montre que les décompositions héritent aussi de l'excellente incrémentalité des algorithmes standards de consistance locale utilisés sur les contraintes d'arité bornée.

5.2 Nonogrammes

Le problème `prob012` de la CSPLib est un puzzle logique NP-complet dans lequel les cases d'une grille doivent être colorées en blanc ou noir. Cette coloration doit respecter la description de la grille, qui spécifie pour chaque ligne et colonne la longueur de chaque segment noir.

Un nonogramme $n \times n$ peut être représenté en utilisant n^2 variables booléennes x_{ij} qui spécifient la couleur de la case en position (i, j). La contrainte sur les longueurs des segments dans chaque ligne et colonne est exprimée par une contrainte globale REGULAR. Nous avons fait deux types d'expériences sur ces nonogrammes pour évaluer le filtrage des fonctions de coût décomposables.

Les Nonogrames assouplis peuvent être construits à partir de nonogrammes classiques en autorisant la violation de la règle des longueurs des segments noirs. Pour cela nous relaxons les contraintes REGULAR sur chaque ligne et chaque colonne en utilisant une mesure de violation de type "distance de Hamming". Le coût associé indique alors combien de cases doivent être modifiées pour satisfaire la description donnée. Ce problème contient $2n$ fonctions de coût globales WEIGHTEDREGULAR avec des portées se chevauchant. Pour pouvoir appliquer le Théoreme 2 sur chacune de ces fonctions de coût globales on doit trouver un ordre des variables qui est un ordre topologique pour chacune de ces fonctions de coût. Il est facile de produire un tel ordre sur ces problèmes. Les variables x_{ij} peuvent, par exemple, être ordonnées en ordre lexicographique, d'en haut à gauche à en bas à droite,c les variables additionnelles étant insérées n'importe où entre leurs variables originales associées. Les portées des fonctions de coût globales sont souvent le reflet de propriétés définies sur le temps (comme dans les problèmes de planning d'infirmières) ou sur l'espace (comme dans les nonogrammes ou les problèmes de traitement de textes). Dans tous ces cas, l'ordre global induit par le temps ou l'espace définit un ordre des variables qui satisfait souvent les conditions du Théoreme 2.

Pour chaque instance de nonogramme aléatoire $n \times n$, on tire de façon uniforme un nombre de segments entre 1 et $\lfloor \frac{n}{3} \rfloor$ pour chaque ligne et colonne. La longueur de chaque segment est tirée de façon uniforme entre 1 et la longueur maximum que permet la place restante et le nombre de segments restants à placer dans cette ligne ou colonne (en considérant une longueur minimale de 1).

Nous avons résolu ces problèmes avec `toulbar2` comme précédemment et avons mesuré le pourcentage de problèmes résolus ainsi que le temps cpu moyen (les problèmes pas encore résolus au bout d'une heure sont comptés pour une heure) sur des échantillons de 100 instances.

Size	Monolithique		Décomposé	
	Résolus	Temps	Résolus	Temps
6×6	100%	1.98	100%	0.00
8×8	96%	358	100%	0.52
10×10	44%	2,941	100%	30.2
12×12	2%	3,556	82%	1,228
14×14	0%	3,600	14%	3,316

Dans ce contexte plus réaliste impliquant plusieurs fonctions de coût globales qui interagissent, les décompositions sont à nouveau, et de loin, l'approche la plus efficace.

Images avec bruit blanc : On génère une grille solution aléatoire avec chaque case colorée en noir avec une probabilité de 0,5. Une instance de nonogramme est créée à partir de la longueur des segments observés dans cette grille aléatoire. Ces instances ont habituellement plusieurs solutions, parmi lesquelles la grille originale. Nous associons à chaque case un coût unaire aléatoire tiré uniformément entre 0 et 99. Ces coûts représentent le prix du coloriage de la case. Une solution de coût minimum est cherchée. Ce problème a été modélisé en `choco` (version. 2.1.3, options par défaut) et `toulbar2` (option `-h:`) en utilisant $2n$ contraintes globales REGULAR. Dans le modèle `choco`, une contrainte SCALAR impliquant toutes les variables est utilisée pour exprimer le critère à optimiser. Dans `toulbar2`, les coûts de coloriage sont exprimés par des fonctions de coût unaires et les contraintes REGULAR sont représentées par des fonctions de coût WEIGHTEDREGULAR avec les poids dans $\{0, k\}$. La version monolithique a été essayée mais donnait des résultats très mauvais.

Nous avons mesuré le pourcentage d'instances résolues et le temps cpu moyen (les problèmes pas encore résolus au bout de demi-heure sont comptés pour une demi-heure) sur des échantillons de 50 instances.

Size	`choco`		`toulbar2`	
	Résolus	Temps	Résolus	Temps
20×20	100%	1.88	100%	0.93
25×25	100%	14.78	100%	3.84
30×30	96%	143.6	96%	99.01
35×35	80%	459.9	94%	218.2
40×40	46%	1,148	66%	760.8
45×45	14%	1,627	32%	1.321

Sur cette classe de problèmes, appliquer la cohérence souple sur les fonctions de coût globales décomposées est préférable au traditionnel filtrage par borne/GAC cohérence d'un pur modèle CP avec variables de coût. Avec les décompositions, l'utilisation directe des fil-

trages par cohérence molle tels que EDAC, qui implique T-DAC, permet une meilleure exploitation des coûts, avec un effort minime d'implantation.

6 Au delà des fonctons de coût décomposables

Il arrive qu'un problème contienne des fonctions de coût globales non décomposables, c'est à dire que toute décomposition en fonctions de coût d'arité bornée est de taille non polynomiale. Cependant, si le réseau de la décomposition est Berge-acyclique, le Théorème 2 s'applique quand même. Avec de tels réseaux de taille exponentielle, le filtrage prendra un temps exponentiel mais produira de bonnes bornes inférieures. La contrainte globale $\sum_{i=1}^{n} a_i x_i = b$ (a et b sont de petits coefficients entiers) peut facilement être décomposée en introduisant $n-3$ variables additionnelles q_i et des contraintes ternaires de somme de la forme $q_{i-1}+a_i x_i = q_i$ avec $i \in [3, n-2]$ et $a_1 x_1 + a_2 x_2 = q_2$, $q_{n-2} + a_{n-1} x_{n-1} + a_n x_n = b$. Les variables additionnelles q_i ont b valeurs, ce qui est exponentiel en la représentation de b. Nous considérons le problème de *Market Split* comme il est défini dans [10, 26]. Le but est de minimiser $\sum_{i=1}^{n} o_i x_i$ tel que $\sum_{i=1}^{n} a_{i,j} x_i = b_j$ pour chaque $j \in [1, m]$ et x_i sont des variables booléennes (o, a et b sont des coefficients entiers positifs). Nous comparons la décomposition Berge-acyclique dans `toulbar2` et une application directe de `cplex` (version 12.2.0.0) sur le programme linéaire en nombres entiers. Nous avons généré des instances aléatoires avec des coefficients entiers choisis aléatoirement dans $[0, 99]$ pour o et a. $b_j = \lfloor \frac{1}{2} \sum_{i=1}^{n} a_{i,j} \rfloor$. Nous avons utilisé 50 instances avec $m = 4, n = 30$, ce qui donne $\max b_j = 918$. Le nombre moyen de noeuds explorés par `cplex` est 50% plus grand qu'avec `toulbar2`. Mais `cplex` était en moyenne 6 fois plus rapide que `toulbar2` sur ces problèmes. Ce problème de sac à dos 0/1 représente probablement un cas très défavorable à `toulbar2`, étant donné que `cplex` contient l'essentiel de ce qui est connu sur la résolution de problèmes de sac à dos en 0/1 (mais seul une partie de ces résultats s'étend à des domaines plus complexes). Il y a deux voies d'amélioration possible pour `toulbar2` sur ces problèmes. La première serait d'utiliser une combinaison des m contraintes de sac à dos en une seule, comme suggéré dans [26]. Une autre consisterait à exploiter directement les propriétés des contraintes linéaires ternaires pour profiter d'une représentation plus compacte du problème et d'un filtrage plus efficace.

7 Travaux connexes

Il faut remarquer que T-DAC est très proche de la notion de "mini-buckets" [12] et de ce fait, le Théorème 2 peut être aisément adapté pour s'appliquer dans ce cadre. Les 'mini-bucket' appliquent une forme limitée d'élimination de variable : quand une variable x est éliminée, les fonctions de coût qui relient x aux variables restantes sont partitionnées dans des ensembles de fonctions de coûts qui contiennent au plus i variables dans leur portée et au plus m fonctions. Si l'on calcule les 'mini-bucket' avec le même ordre de variables que T-DAC, avec $m = 1$ et une valeur de i non bornée, on obtient la même fonction de coût marginale que T-DAC sur la variable racine r, avec la même complexité temporelle. Les 'mini-bucket' peuvent être mis en œuvre de deux façons différentes : la variante statique ne nécessite aucune mise à jour durant la recherche mais restreint l'ordre d'instanciation des variables à un ordre statique ; la variante dynamique permet d'utiliser un ordre d'affectation des variables dynamique (DVO) mais souffre d'un manque d'incrémentalité. Les cohérences locales souples, du fait qu'elles s'appuient sur la notion d'EPT, fournissent toujours un problème équivalent, offrant ainsi une incrémentalité totale durant la recherche ainsi qu'une compatibilité parfaite avec les ordres d'affectation dynamique des variables. Les cohérences d'arc souples offrent également une complexité spatiale en $O(edr)$ alors que les 'mini-bucket' peuvent demander un espace exponentiel en i.

8 Conclusion

Dans ce papier, nous avons étendu la décomposition de contraintes aux fonctions de coût apparaissant dans des CFN. Pour des fonctions de coût ayant une décomposition Berge-acyclique, nous avons montré qu'un filtrage simple tel que la cohérence d'arc directionnelle, fournit un filtrage sur la décomposition comparable à celui de la fonction de coût globale elle-même (à condition de fournir un ordre idoine pour les variables). Les résultats que nous obtenons pour le filtrage fourni par la cohérence d'arc virtuelle (plus fort que DAC) sont identiques et ne nécessitent pas d'ordre particulier.

L'application de ces résultats sur la classe triviale des contraintes globales Berge-acycliques ou *fonctions de coût Berge-acycliques décomposables*, est déjà très significative, car elle permet d'établir des cohérences locales souples sur des réseaux contenant des fonctions de coût Berge-acycliques décomposables telles que REGULAR, GRAMMAR, AMONG,...

Nous avons montré que pour ces contraintes globales Berge-acycliques peuvent aussi être relaxées en des

fonctions de coût globales Berge-acycliques généralisant la mesure de violation basée sur la « décomposition ». Ce résultat fournit une longue liste de fonctions de coût Berge-acycliques décomposables. Nos résultats expérimentaux basés sur l'utilisation de T-DAC sur la relaxation de la contrainte REGULAR en la fonction de coût WEIGHTEDREGULAR montre que cette approche basée sur la décomposition permet une importante diminution du temps de calcul par rapport au propagateur monolithique. De plus, l'utilisation de telles fonctions de coût simplifie grandement l'implémentation par rapport aux versions monolithiques de ces fonctions de coût.

Afin d'évaluer expérimentalement l'intérêt pratique de l'utilisation de VAC, il est nécessaire d'implémenter techniquement VAC sur des fonctions de coût non binaires.

Bien que les travaux présentés ici soient restreint aux décompositions Berge-acycliques, ceux-ci ouvrent la voie vers une forme plus générale de « décomposition structurelle » des fonctions de coût globales décomposables en une structure acyclique composée de fonctions de coût locales, possédant des séparateurs de taille bornée (pas nécessairement de taille 1). Ces fonctions de coût globales décomposées pourraient alors être filtrées efficacement en utilisant des transformations préservant l'équivalence (dédiées et incrémentales) basées sur des algorithmes issus de la programmation dynamique.

Références

[1] C. Beeri, R. Fagin, D. Maier, and M.Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30 :479–513, 1983.

[2] N. Beldiceanu, M. Carlsson, R. Debruyne, and T. Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4) :339–362, 2005.

[3] C. Bessiere. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3. Elsevier, 2006.

[4] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide : A useful special case of the cardpath constraint. In *Proc. of ECAI'08*, pages 475–479, 2008.

[5] C. Bessiere and P. Van Hentenryck. To be or not to be ... a global constraint. In *Proc. CP'03*, pages 789–794, 2003.

[6] E. Boros and P. Hammer. Pseudo-Boolean Optimization. *Discrete Appl. Math.*, 123 :155–225, 2002.

[7] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174 :449–478, 2010.

[8] M C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342, 2003.

[9] M C. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2) :199–227, 2004.

[10] Gérard Cornuéjols and Milind Dawande. A class of hard small 0-1 programs. In *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, Houston, Texas, USA, June 22-24, 1998, Proceedings*, pages 284–293, 1998.

[11] Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *MFCS*, volume 711 of *Lecture Notes in Computer Science*, pages 392–402. Springer, 1993.

[12] Rina Dechter. Mini-buckets : A general scheme for generating approximations in automated reasoning. In *Proc. of the* 16^{th} *IJCAI*, pages 1297–1303, 1997.

[13] George Katsirelos, Nina Narodytska, and Toby Walsh. The weighted grammar constraint. *Annals OR*, 184(1) :179–207, 2011.

[14] D. Koller and N. Friedman. *Probabilistic graphical models.* MIT press, 2009.

[15] J. Larrosa, S. de Givry, F. Heras, and M. Zytnicki. Existential arc consistency : getting closer to full arc consistency in weighted CSPs. In *Proc. of the* 19^{th} *IJCAI*, pages 84–89, Edinburgh, Scotland, August 2005.

[16] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of the* 18^{th} *IJCAI*, pages 239–244, Acapulco, Mexico, August 2003.

[17] Javier Larrosa and Thomas Schiex. Solving weighted CSP by maintaining arc consistency. *Artif. Intell.*, 159(1-2) :1–26, 2004.

[18] JHM. Lee and KL. Leung. Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *Journal of Artificial Intelligence Research*, 43 :257–292, 2012.

[19] Jimmy Lee and K. L. Leung. A stronger consistency for soft global constraints in weighted constraint satisfaction. In Maria Fox and David Poole, editors, *Proc. of AAAI'10*. AAAI Press, 2010.

[20] Jimmy Ho-Man Lee and Ka Lun Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In Craig Boutilier, editor, *Proc of the* 21^{th} *IJCAI*, pages 559–565, 2009.

[21] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3) :425–440, 1991.

[22] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

[23] Thierry Petit, Jean-Charles Régin, and Christian Bessiere. Specific filtering algorithms for over-constrained problems. In *CP*, pages 451–463, 2001.

[24] Martí Sánchez, Simon de Givry, and Thomas Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1-2) :130–154, 2008.

[25] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems : hard and easy problems. In *Proc. of the* 14^{th} *IJCAI*, pages 631–637, Montréal, Canada, August 1995.

[26] M. A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118(1-4) :73–84, 2003.

[27] Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming : Flow-based soft global constraints. *J. Heuristics*, 12(4-5) :347–373, 2006.

[28] M. Zytnicki, C. Gaspin, S. de Givry, and T. Schiex. Bounds arc consistency for weighted CSPs. *Journal of Artificial Intelligence Research*, 35(2) :593–621, 2009.

[29] M. Zytnicki, C. Gaspin, and T. Schiex. A new local consistency for weighted CSP dedicated to long domains. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)*, pages 394–398, Dijon, France, April 2006.