

# Cohérence d'arc virtuelle pour les CSP pondérés

M. Cooper  
IRIT, UPS  
Toulouse  
cooper@irit.fr

S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki  
INRA, UR 875  
Castanet Tolosan  
{degivry,tschiex}@toulouse.inra.fr

## Résumé

L'optimisation d'une combinaison de fonctions de coût définies sur des variables discrètes est un problème central dans de nombreux formalismes tels que les réseaux probabilistes, le problème de satisfiabilité maximum, les réseaux de contraintes pondérés (WCSP) ou les graphes de facteurs. De récents résultats ont montré que la maintenance d'une forme de cohérence locale dans un algorithme de séparation-évaluation (Branch and Bound) fournit des minorants qui sont assez puissants pour résoudre de nombreuses instances réelles.

Nous présentons ici la cohérence d'arc virtuelle (Virtual Arc Consistency, VAC) qui planifie et applique itérativement des séquences d'opération de propagation de coûts fractionnaires qui garantissent de transformer un WCSP en un autre WCSP équivalent incorporant un minorant accru. Bien que plus faible que l'arc cohérence optimale (OSAC) récemment proposée, VAC est plus rapide et est capable de résoudre le langage polynomial défini par des fonctions de coût sous-modulaires. Le maintien de VAC pendant la recherche conduit à d'importantes améliorations sur des problèmes difficiles de grande taille et nous a permis de clore deux instances bien connues du problème d'affectation de fréquence.

## Abstract

Optimizing a combination of local cost functions on discrete variables is a central problem in many formalisms such as in probabilistic networks, maximum satisfiability, weighted CSP or factor graphs. Recent results have shown that maintaining a form of local consistency in a Branch and Bound search provides bounds that are strong enough to solve many practical instances.

In this paper, we introduce Virtual Arc Consistency (VAC) which iteratively computes and applies *sequences* of cost propagation over rational costs that are guaranteed to transform a WCSP in another WCSP with an improved constant cost. Although not as strong as Optimal Soft Arc Consistency, VAC is faster and powerful enough to solve submodular problems. Maintaining VAC inside branch and bound leads to important im-

provements in efficiency on large difficult problems and allowed us to close two famous frequency assignment problem instances.

## 1 Introduction

L'analyse des modèles graphiques est un problème central en IA. L'optimisation du coût combiné de fonctions de coût locales, centrale dans le cadre des CSP valués [16], capture des problèmes tels que le problème de satisfiabilité maximum (MaxSAT), la résolution des CSP pondérés ou le problème de recherche d'une explication de probabilité maximum dans les réseaux probabilistes. Il a des applications en *allocation de ressource*, en *enchères combinatoires*, en *bioinformatique*...

Les approches de type "programmation dynamique" basées sur l'élimination de variables ou par arbre de jonction ont été largement mobilisées pour résoudre de tels problèmes mais elles sont intrinsèquement limitées par leur complexité exponentielle en temps et en espace lors de leur application à des modèles graphiques de grande largeur d'arbre. Au contraire, les approches de type "séparation-évaluation" permettent de conserver une complexité spatiale raisonnable mais nécessitent de bons minorants (forts et peu coûteux) pour avoir une efficacité correcte.

Durant ces dernières années, des minorants de qualité croissante, basés sur l'établissement de cohérence locale pour les CSP pondérés, ont été définis. Ces cohérences locales sont établies via l'application répétée de "transformations préservant l'équivalence" (Equivalence Preserving Transformations ou EPT [6]) qui étendent les opérations de cohérence locale utilisées dans les CSP classiques. Une tendance similaire a suivi dans le cadre MaxSAT dans lequel des "règles d'inférence" préservant l'équivalence sont maintenant utilisées dans les solveurs les plus récents [9]. Ces EPT dé-

placent des coûts entiers entre des fonctions de coût de portée différente et peuvent éventuellement permettre d’augmenter la fonction de coût de portée vide (une constante) jusqu’à une valeur non triviale. Cette valeur fournit un minorant évident qui peut être maintenu durant une recherche arborescente.

Au niveau de la cohérence d’arc, l’application non restreinte chaotique d’EPT ne converge généralement pas vers un point fixe unique [15] et peut ne pas terminer. Différentes restrictions heuristiques qui terminent toujours ont donc été introduites, conduisant à différentes variantes de l’arc cohérence telles que AC\*, DAC\*, FDAC\*, EDAC\* [8]... La définition récente de l’arc cohérence optimale (Optimal Soft Arc Consistency ou OSAC [5]) à montré qu’il était possible de pré-calculer, en temps polynomial, un *ensemble* d’EPT déplaçant des coûts rationnels et maximisant la valeur du minorant défini par la fonction de coût de portée vide. Cet algorithme, basé sur le programmation linéaire, fournit des minorants forts mais qui semblent trop coûteux pour être maintenus durant la recherche.

Dans cet article, nous présentons l’arc cohérence virtuelle (Virtual Arc Consistency, VAC) qui utilise un algorithme d’établissement de la cohérence d’arc classique pour produire des *séquences* d’EPT à coût rationnel qui augmentent toujours le minorant. Bien que moins puissante qu’OSAC, VAC est rapide à établir et fournit des minorants assez forts pour résoudre les fonctions de coût sous-modulaires (comme OSAC). C’est l’un des ingrédients essentiels qui nous a permis de clore deux instances difficiles d’affectation de fréquence qui sont restées ouvertes pendant plus de 10 années.

## 2 Préliminaires

Un CSP pondéré (Weighted CSP, WCSP) est un quadruplet  $(X, D, W, m)$ .  $X$  et  $D$  sont des ensembles de  $n$  variables et domaines, comme dans les CSP classiques. Le domaine de la variable  $i$  est noté  $D_i$ . Étant donné un sous-ensemble de variables  $S \subseteq X$ , on note  $\ell(S)$  l’ensemble des  $n$ -uplets (tuples) définis sur  $S$ .  $W$  est un ensemble de fonctions de coût. Chaque fonction de coût (ou contrainte molle)  $w_S$  de  $W$  est définie sur un ensemble de variables  $S$  appelé sa portée et supposé différent pour chaque fonction de coût. Une fonction de coût  $w_S$  assigne un coût à chaque affectation des variables de  $S$  i.e. :  $w_S : \ell(S) \rightarrow [0, m]$ . L’ensemble des coûts possibles est  $[0, m]$  où  $m \in \{1, \dots, +\infty\}$  représente un coût intolérable. Les coûts sont combinés par l’addition bornée  $\oplus$ , définie par  $a \oplus b = \min\{m, a + b\}$ , et comparés entre eux via  $\geq$ . Notez que le coût intolérable  $m$  peut être fini ou infini et qu’un coût  $b$  peut être soustrait d’un coût  $a$  plus large en utilisant l’opé-

ration  $\ominus$  où  $a \ominus b$  est égal à  $(a - b)$  si  $a \neq m$  et à  $m$  sinon.

Pour les fonctions de coût binaires et unaires, nous utilisons des notations simplifiées : une fonction de coût binaire entre les variables  $i$  et  $j$  est notée  $w_{ij}$ . Une fonction de coût unaire sur la variable  $i$  est notée  $w_i$ . Nous faisons l’hypothèse qu’il existe une fonction de coût unaire  $w_i$  pour chaque variable ainsi qu’une fonction de coût d’arité nulle (une constante) notée  $w_\emptyset$ . Si  $m = 1$ , notez que le WCSP est équivalent au problème classique CSP (un coût de 1 étant associé aux combinaisons de valeurs interdites). Pour rendre évident que dans ce cas les fonctions de coûts ne représentent que de simples contraintes, elles seront alors notées  $c_S$  au lieu de  $w_S$ .

Le coût d’une affectation complète  $t \in \ell(X)$  dans un problème  $P = (X, D, W, m)$  est égal à  $Val_P(t) = \bigoplus_{w_S \in W} w_S(t[S])$  où  $t[S]$  représente la projection habituelle d’un tuple sur l’ensemble de variables  $S$ . La minimisation de  $Val_P(t)$  définit un problème d’optimisation avec un problème de décision associé qui est NP-complet.

L’établissement d’une cohérence locale donnée sur un problème  $P$  consiste à transformer  $P = (X, D, W, m)$  en un problème  $P' = (X, D, W', m)$  qui est équivalent à  $P$  ( $Val_P = Val_{P'}$ ) et qui satisfait la propriété de cohérence locale considérée. Ce processus peut augmenter  $w_\emptyset$  et fournir ainsi un minorant amélioré du coût optimum. Il s’appuie sur l’application de transformations préservant l’équivalence (EPT) qui déplace les coûts entre des portées différentes.

L’algorithme 1 décrit deux EPT élémentaires. **Project()** s’applique dans la portée d’une fonction de coût  $w_S$ . Elle déplace une quantité de coût  $\alpha$  de  $w_S$  vers une fonction de coût unaire  $w_i, i \in S$ , sur une valeur  $a \in D_i$ . Si le coût  $\alpha$  est négatif, cela signifie que le coût est déplacé de la fonction de coût unaire  $w_i$  vers la fonction de coût  $w_S$  : on parle alors d’extension. Afin d’éviter l’apparition de coûts négatifs dans  $P'$ , il est nécessaire que  $-w_i(a) \leq \alpha \leq \min_{t \in \ell(S), t[\{i\}] = a} \{w_S(t)\}$ . De façon similaire, **UnaryProject()** s’applique sur un sous-problème défini par une seule variable  $i \in X$ . Elle déplace un coût  $\alpha$  de la fonction de coût unaire  $w_i$  vers le fonction d’arité nulle  $w_\emptyset$  (avec  $-w_\emptyset \leq \alpha \leq \min_{a \in D_i} \{w_i(a)\}$  de façon à conserver des coûts positifs).

Un WCSP binaire est noeud-cohérent (NC) ssi  $\forall i \in X, \exists a \in D_i, w_i(a) = 0$  ( $(i, a)$  est appelé un support unaire pour  $i$ ). Il vérifie la cohérence d’arc (AC) s’il est NC et si  $\forall i \in X, \forall w_{ij} \in W$  et  $\forall a \in D_i, \exists b \in D_j$  tel que  $w_{ij}(a, b) = 0$  ( $(j, b)$  est appelé un support de  $a$  sur  $w_{ij}$ ). Quand une valeur  $(i, a)$  n’a pas de support sur  $w_{ij}$ , on peut en créer un en appliquant la transformation **Project** sur la fonction de coût  $w_{ij}$  vers la valeur  $(i, a)$  avec

le coût le plus grand possible. Ceci est montré dans la figure 1a. Ce CSP pondéré a deux variables avec deux valeurs  $a$  et  $b$ . Les sommets, représentant des valeurs, peuvent être pondérés par les coûts unaires. Une arête connectant deux valeurs représente un coût binaire de 1. Si deux valeurs ne sont pas connectées, le coût associé est implicitement nul. Notez que la valeur  $(1, b)$  n'a pas de support sur la variable 2. On peut appliquer la transformation  $\text{Project}(w_{12}, 1, b, 1)$  qui crée un coût unaire de 1 sur  $(1, b)$ . L'application de la transformation  $\text{UnaryProject}(1, 1)$  sur le réseau résultant (figure 1b) rend le problème AC et augmente  $w_\emptyset$  de 1. Différents ordres d'application des transformations peuvent conduire à des points fixes (fermetures) différents : en considérant d'abord l'absence de support de  $(2, a)$  sur  $w_{12}$  et en appliquant  $\text{Project}(w_{12}, 2, a, 1)$ , on aboutit au réseau de la figure 1c. Il est AC, mais  $w_\emptyset$  vaut 0.

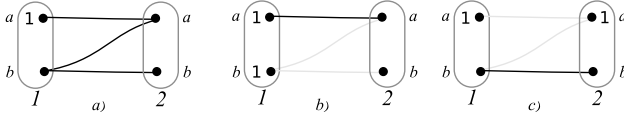


FIG. 1 – Un CSP pondéré et deux fermetures AC.

Les niveaux de cohérence locale existants (AC, DAC, FDAC, EDAC) peuvent être vus comme des heuristiques efficaces cherchant à s'approcher d'une fermeture arc cohérente optimale (maximisant  $w_\emptyset$ ). En utilisant des coûts rationnels, il a été montré qu'une telle fermeture arc cohérente peut être atteinte via l'établissement de la cohérence d'arc optimale (OSAC, [5]). Ceci nécessite la résolution d'un problème linéaire de grande taille et son utilisation a donc été limitée au prétraitement. Nous proposons dans la section suivante un mécanisme alternatif plus efficace basé sur un algorithme de cohérence d'arc classique et qui peut être maintenu durant la recherche.

---

**Algorithme 1** : Les transformations élémentaires

---

**Procédure**  $\text{Project}(w_S, i, a, \alpha)$

$w_i(a) \leftarrow w_i(a) \oplus \alpha;$   
**pour tous les**  $(t \in \ell(S)$  tel que  $t[\{i\}] = a)$  **faire**  
 $w_S(t) \leftarrow w_S(t) \ominus \alpha;$

**Procédure**  $\text{UnaryProject}(i, \alpha)$

$w_\emptyset \leftarrow w_\emptyset \oplus \alpha;$   
**pour tous les**  $(a \in D_i)$  **faire**  $w_i(a) \leftarrow w_i(a) \ominus \alpha;$

---

### 3 Cohérence d'arc virtuelle

Étant donné un WCSP  $P = (X, D, W, m)$ , on définit  $\text{Bool}(P)$  comme le CSP classique  $(X, D, \overline{W})$  où  $c_S \in \overline{W}$  si et seulement si  $\exists w_S \in W$  et  $S \neq \emptyset$  tel que  $\forall t \in \ell(S)$  ( $t \in c_S \Leftrightarrow w_S(t) = 0$ ).  $\text{Bool}(P)$  est un CSP classique dont les solutions (si il y en a) ont un coût égal à  $w_\emptyset$  dans  $P$ .

**Définition 1** *Un WCSP satisfait la cohérence d'arc virtuelle (VAC) si la fermeture arc cohérente classique de  $\text{Bool}(P)$  n'est pas vide.*

Si un WCSP  $P$  ne satisfait pas la propriété VAC, alors  $\text{Bool}(P)$  est incohérent et on sait que les solutions de  $P$  ont un coût strictement plus grand que  $w_\emptyset$ . De façon plus intéressante, on va voir qu'en simulant le filtrage par cohérence d'arc classique de  $\text{Bool}(P)$ , il est possible de construire une séquence de transformations préservant l'équivalence (EPT) et qui mène prouvément à une augmentation de  $w_\emptyset$ . Considérons par exemple le problème de la figure 2(a). Ce problème est une WCSP booléen binaire que l'on peut interpréter comme un problème de type Max-SAT défini par les clauses  $\bar{x}; x \vee \bar{y}; x \vee z; y \vee \bar{z}$ . Il satisfait la propriété EDAC. Notez que  $\text{Bool}(P)$  est représenté par la même figure dès lors que l'on fait l'hypothèse que  $m = 1$  (un coût de 1 représente une combinaison interdite).

Dans une première phase, on filtre le problème  $\text{Bool}(P)$  par cohérence d'arc classique. La fermeture arc cohérente obtenue est décrite dans la figure 2(b) : comme la valeur  $(x, t)$  est interdite, les valeurs  $(y, t)$  et  $(z, f)$  n'ont pas de support sur  $x$  et peuvent être effacées (marquées avec un coût de 1). Pour chaque effacement, nous mémorisons la source de l'effacement par une flèche grise pointant vers la variable n'offrant pas de support. La valeur  $(z, t)$  peut être effacée du fait de l'effacement de  $(y, t)$  et le domaine de  $z$  se retrouve réduit à l'ensemble vide (wipe-out). Du fait que le WCSP d'origine a des coûts tous entiers, on pourrait déduire un minorant de 1 pour le problème original  $P$ , mais sans disposer d'une version transformée équivalente de  $P$ .

On met donc en oeuvre une seconde phase qui retrace les étapes de la première phase en partant de la variable dont le domaine a été épuisé. Faisons l'hypothèse qu'une quantité inconnue de coût  $\lambda$  va pouvoir être déplacée vers  $w_\emptyset$  à partir de la variable  $z$  dont le domaine est vide. Pour ce faire, des coûts de au moins  $\lambda$  doivent être disponibles à chacune des valeurs de la variable  $z$  afin de les projeter sur  $w_\emptyset$ . En suivant les causes d'effacement, nous savons que ces coûts peuvent être obtenus par projection à partir des fonctions de coût  $w_{yz}$  et  $w_{xz}$  respectivement. Les coûts  $w_{yz}(f, t)$  et  $w_{xz}(f, f)$  étant non nuls dans  $P$ , il sera

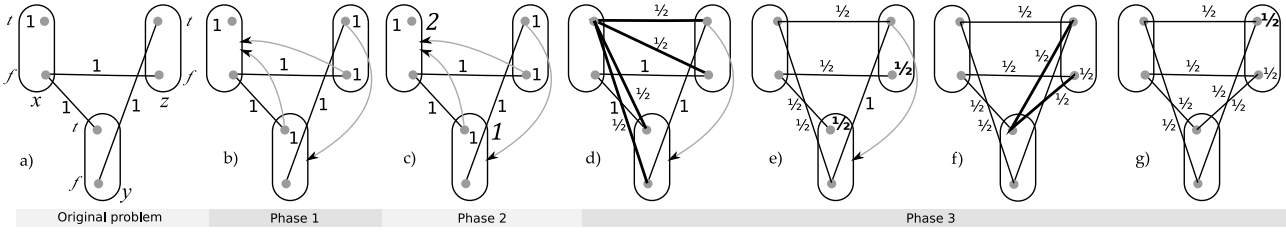


FIG. 2 – Un WCSP où VAC est plus puissant qu'EDAC.

possible de déplacer les coûts à partir de là, et il est donc inutile de remonter plus loin. Les autres coûts nécessaires doivent être extraits des fonctions de coût  $w_y$  et  $w_x$  par extension. Une quantité de coût de  $\lambda$  doit être obtenue récursivement via  $w_{xy}$  vers  $w_x$ . Le processus s'arrête quand tous les coûts requis sont non-nuls dans le WCSP original  $P$ . Nous sommes alors capables de dénombrer le nombre de demandes d'une quantité de coût  $\lambda$  sur chacun des coûts non nuls identifiés. Ces comptes sont montrés en italique sur la figure 2(c). Ici, le nombre maximum de demandes de coût est atteinte sur  $w_x(t)$  avec 2 demandes. Étant donné que  $w_x(t) = 1$  dans  $P$ , la quantité maximum de coût que l'on peut affecter à  $\lambda$  est donc  $\frac{1}{2}$ .

Dans une troisième phase, on applique directement toutes les transformations identifiées dans la phase 2, dans leur ordre inverse et en utilisant la valeur maximum de  $\lambda = \frac{1}{2}$  identifiée. Le processus est illustré dans les figures 2(d) à 2(g) dans lesquelles les extensions et projections de coût réalisées sont montrées en gras. Le WCSP final obtenu est équivalent à notre problème d'origine mais possède un minorant  $w_\emptyset = \frac{1}{2}$ . Étant équivalent au problème original, le processus (ou tout autre traitement) peut être à nouveau appliqué. Ici, la fermeture arc cohérente de du nouveau  $\text{Bool}(P)$  n'est pas vide et le problème obtenu satisfait donc la propriété de cohérence d'arc virtuelle (VAC). Le problème original étant à coûts entiers, on dispose en fait d'un minorant de 1 sur le coût d'une solution optimale.

Le théorème qui suit montre que si le filtrage par cohérence d'arc de  $\text{Bool}(P)$  produit une fermeture vide, il est alors possible d'augmenter  $w_\emptyset$  par une séquence de transformations préservant l'équivalence au niveau arc (et réciproquement).

**Théorème 1** *Soit  $P$  un WCSP. Il existe une séquence de transformations préservant l'équivalence au niveau arc qui, une fois appliquées à  $P$  produisent un WCSP avec une augmentation de la fonction de coût  $w_\emptyset$  si et seulement si la fermeture arc cohérente de  $\text{Bool}(P)$  est vide.*

Preuve :  $\Rightarrow$  : Soit  $O_1, \dots, O_t$  une séquence d'EPT au niveau arc dans  $P$  qui produisent un WCSP équivalent avec un  $w_\emptyset$  accru. Soit  $O'_1, \dots, O'_t$  les EPT correspon-

dantes avec un coût projeté ou étendu fixé à 1. L'application de cette séquence d'opérations à  $\text{Bool}(P)$  (vu comme un WCSP avec  $m = 1$ ) effectue un filtrage par cohérence d'arc de  $\text{Bool}(P)$  qui conduit inévitablement à un domaine vide.

$\Leftarrow$  : Soit  $O_1, \dots, O_t$  une séquence d'EPT dans  $\text{Bool}(P)$  qui mène à un problème ayant un domaine vide. Sans perte de généralité, on peut faire l'hypothèse qu'aucune paire d'opérations  $O_i, O_j$  n'est strictement identique car les mêmes EPT n'ont jamais besoin d'être appliquées deux fois dans un CSP classique (idempotence de  $\oplus$ ). Chaque opération  $O_i$  correspond soit à une opération de type **Project** soit à une opération de type **UnaryProject** dans  $\text{Bool}(P)$  vu comme un WCSP avec  $m = 1$ . Soit  $O'_i$  l'EPT correspondante dans  $P$  appliquée avec un coût de  $\delta/e^i$ , où  $\delta$  est le coût non nul le plus petit qui apparaît dans  $P$ . On divise le coût par  $e = |W|$  à chaque opération car un coût peut, dans le pire des cas, être divisé en quantités plus petites qui doivent être étendues vers toutes les fonctions de coût impliquant la variable (ou projeté vers toutes les variables dans la portée d'une même fonction de coût). Même dans le cas d'une division maximale, le coût  $\delta/e^i$  peut toujours être déplacé lors de l'application de l'opération  $O'_i$ . Après l'application de  $O'_1, \dots, O'_t$  à  $P$ , on obtient nécessairement par projection sur  $w_\emptyset$  une augmentation de  $w_\emptyset$  plus grande que  $\delta/e^t > 0$ .  $\square$

Il est facile de montrer que VAC est plus fort que la cohérence d'arc existentielle (EAC) et elle peut même résoudre les problèmes sous-modulaires, un langage polynomial non trivial des CSP pondérés [4]. Si l'on fait l'hypothèse que chaque domaine est ordonné, une fonction de coût  $w_S$  est sous-modulaire si et seulement si  $\forall t, t' \in \ell(S), w(\max(t, t')) + w(\min(t, t')) \leq w(t) + w(t')$  où  $\max$  et  $\min$  représentent l'application point à point de  $\max$  (resp.  $\min$ ) sur les valeurs de  $t, t'$ . Cette classe inclut des fonctions de coût telles que  $\sqrt{x^2 + y^2}$  ou  $(x \geq y ? (x - y)^r : m)$  avec  $(r \geq 1)$ , utile en bioinformatique [20] et permettant de capturer une forme de CSP temporel simple avec des préférences linéaires [11].

**Théorème 2** *Soit  $P$  un WCSP dont les fonctions de*

coût sont toutes sous-modulaires et qui satisfait la propriété de cohérence d'arc virtuelle. Alors, une solution optimale de  $P$  peut être trouvée en un temps polynomial et son coût est égal à  $w_\emptyset$ .

Schéma de preuve : dans  $\text{Bool}(P)$ , la définition de la sous-modularité se réécrit  $c(t) \wedge c(t') \Rightarrow c(\max(t, t')) \wedge c(\min(t, t'))$  indiquant que les relations de  $\text{Bool}(P)$  sont à la fois fermées pour max (max-closed) et pour min. Étant donné que le WCSP satisfait VAC, la fermeture arc cohérente de  $\text{Bool}(P)$  n'est pas vide. On sait d'autre part que le filtrage par cohérence d'arc résout les problèmes fermés pour max [10]. Une solution de  $\text{Bool}(P)$  peut donc être produite en temps polynomial (en considérant les valeurs maximum des domaines filtrés). Son coût dans le WCSP original est, par définition de  $\text{Bool}(P)$ , égal à  $w_\emptyset$  et donc optimal.  $\square$

Les transformations `Project` et `UnaryProject` préservant la sous-modularité [7], l'établissement de VAC sur un problème sous-modulaire permet donc de résoudre le problème. L'ordre utilisé sur les domaines n'apparaissant pas dans le fonctionnement de VAC, VAC peut déterminer le coût optimal de problèmes sous-modulaires permutés [17] sans nécessiter de réordonner les domaines.

## 4 Établissement de la cohérence d'arc virtuelle

Dans cette section, nous nous restreignons par souci de simplicité à des WCSP binaires, mais VAC peut être appliqué à des problèmes d'arité arbitraire en s'appuyant sur la cohérence d'arc généralisée (GAC) au lieu d'AC (la structure de donnée `tueur` utilisée dans la suite devient alors une portée de fonction de coût). Comme on l'a vu dans l'exemple précédent, le processus d'établissement de VAC se déroule en trois phases. La première phase consiste à appliquer un algorithme instrumenté de filtrage par AC sur  $\text{Bool}(P)$ . Cet algorithme, représenté par la fonction `Instrumented-AC` n'est pas décrit ici du fait de sa simplicité. Si aucun domaine vide n'apparaît à la fin du filtrage, le problème est déjà VAC et 0 est retourné. Sinon, le numéro de la variable dont le domaine s'est épuisé est retourné. L'instrumentation doit collecter deux types d'information : pour toute valeur  $(i, a)$  effacée du fait de l'absence de support sur une contrainte  $c_{ij}$ , la structure de donnée `tueur`( $i, a$ ) doit contenir la variable  $j$ . De plus, la valeur  $(i, a)$  elle-même doit être poussée dans une pile  $P$ . Ces deux structures de données ont une complexité spatiale en  $O(ed)$  et  $O(nd)$  respectivement et elles ne changent pas les complexités temporelles et spatiales des algorithmes de filtrage par arc cohérence

optimaux.

La seconde phase est décrite par l'algorithme 2. Il exploite la pile  $P$  et la structure `tueur` pour parcourir l'historique des propagations réalisées et construire ainsi un sous-ensemble minimal pour l'inclusion des valeurs effacées suffisant pour expliquer l'épuisement du domaine observé. Pour cela, un booléen  $M(i, a)$  est fixé à la valeur vraie si l'effacement de  $(i, a)$  est nécessaire pour expliquer le wipe-out. Cette phase calcule également la quantité de coût  $\lambda$  qu'il sera finalement possible d'ajouter à  $w_\emptyset$ . En s'appuyant sur la structure `tueur`, il est toujours possible de remonter les causes d'effacement jusqu'à ce qu'un coût non nul soit atteint : ce coût sera la source dont pourra être extrait la quantité  $\lambda$ . Cependant, dans les CSP classiques, un effacement donné peut être la cause directe de plusieurs autres effacements. Pour calculer la valeur de  $\lambda$ , il faut donc calculer combien de fois une source de coût identifiée a été sollicitée dans le WCSP original, que ce soit au niveau unaire ou binaire. Pour un tuple (une paire ou une valeur)  $t_S$  de portée  $S$ , tel que  $w_S(t_S) \neq 0$ , nous utilisons un entier  $k(t_S)$  pour stocker le nombre de demandes de coût de  $\lambda$  sur  $w_S(t_S)$ .

L'utilisation de la pile  $P$  garantit que les valeurs effacées sont explorées dans un ordre anti-causal : une valeur effacée est toujours explorée avant les effacements qui ont pu mener à sa suppression. Ainsi, quand le nombre de demandes de coût pour un tuple donné est calculé, ce calcul est basé sur des coûts déjà établis, et inductivement corrects. *In fine*, il sera possible de calculer  $\lambda$  comme le minimum de  $\frac{w_S(t_S)}{k(t_S)}$  sur tous les  $t_S$  tels que  $k(t_S) \neq 0$ .

Initialement, tous les  $k$  sont initialisés à 0 excepté à la variable  $i_0$  dont le domaine a été épuisé et pour laquelle une demande de coût est associée à chaque valeur afin de pouvoir augmenter  $w_\emptyset$  (ligne 1). Une valeur  $(i, a)$  extraite de  $P$  (ligne 2) a été effacée par manque de support sur la variable `tueur`( $i, a$ ) =  $j$ . Si cet effacement est nécessaire pour expliquer le wipe-out (ligne 3), le manque de support peut être dû au fait que :

1. la paire  $(a, b)$  est interdite par  $c_{ij}$  dans  $\text{Bool}(P)$  ce qui veut dire que  $w_{ij}(a, b) \neq 0$  (ligne 5). Le parcours arrière des effacements peut s'arrêter, le nombre de demande de coût faites sur la paire  $(a, b)$  (ligne 6) et  $\lambda$  (ligne 7) sont alors mis à jour.
2. sinon, la valeur  $(j, b)$  a été effacée et  $k((i, a))$  demandes de coût lui sont transmises. Notez que si différentes valeurs de la variable  $i$  effectuent des demandes de coût sur la valeur  $(j, b)$ , il suffira de répondre à la demande *maximum* car une extension de coût vers  $w_{ij}$  fournit des coûts à tous les  $w_{ij}(a, b)$ . Pour maintenir ce maximum, nous utilisons une autre structure de données, notée

$k_i((j, b))$ , pour stocker le nombre de demandes maximum de coût effectuée par  $i$  sur  $(j, b)$ . On a donc  $k((i, a)) = \sum k_j((i, a))$ . Ici, si le nombre de demandes est supérieur au maximum déjà enregistré (ligne 8),  $k_i((j, b))$  (ligne 9) et  $k((j, b))$  (ligne 10) doivent être mis à jour. Si aucun coût unaire  $w_j(b)$  n'explique l'effacement de  $(j, b)$ , cela signifie que  $(j, b)$  a été effacé par le filtrage par AC et il est nécessaire de poursuivre le parcours de la raison de l'effacement de  $(j, b)$  récursivement (ligne 11). Sinon, ce parcours peut s'arrêter à  $(j, b)$  et  $\lambda$  est mis à jour (ligne 12).

La dernière phase est décrite dans l'algorithme 3. Elle modifie directement le WCSP original en appliquant les transformations identifiées dans la phase précédente, dans l'ordre inverse, via la pile  $R$ . Comme le théorème 1 le montre, le nouveau WCSP aura un  $w_\emptyset$  augmenté de  $\lambda$ .

---

**Algorithme 2** : VAC - Phase 2 : Calcul de  $\lambda$

---

```

Initialiser tous les  $k, k_j$  à 0,  $\lambda \leftarrow m$  ;
 $i_0 \leftarrow \text{Instrumented-AC}()$  ;
si ( $i_0 = 0$ ) alors retourner ;
pour tous les  $a \in D_{i_0}$  faire
1   $k((i, a)) \leftarrow 1, M(i, a) \leftarrow \text{vrai}$  ;
   si ( $w_i(a) \neq 0$ ) alors
   |  $M(i, a) \leftarrow \text{faux}$   $\lambda \leftarrow \min(\lambda, w_i(a))$  ;
tant que ( $P \neq \emptyset$ ) faire
2  ( $i, a) \leftarrow P.Pop()$  ;
3  si ( $M(i, a)$ ) alors
   |  $j \leftarrow \text{tueur}(i, a)$  ;  $R.Push(i, a)$  ;
4  pour tous les  $b \in D_j$  faire
5  | si ( $w_{ij}(a, b) \neq 0$ ) alors
6  | |  $k((i, a), (j, b)) \leftarrow k((i, a), (j, b)) + k((i, a))$  ;
7  | |  $\lambda \leftarrow \min(\lambda, \frac{w_{ij}(a, b)}{k((i, a), (j, b))})$  ;
8  | sinon si ( $k((i, a)) > k_i((j, b))$ ) alors
9  | |  $k_i((j, b)) \leftarrow k((i, a))$  ;
10 | |  $k((j, b)) \leftarrow k((j, b)) + k((i, a)) - k_i((j, b))$  ;
11 | | si ( $w_j(b) = 0$ ) alors  $M(j, b) \leftarrow \text{vrai}$  ;
12 | | sinon  $\lambda \leftarrow \min(\lambda, \frac{w_j(b)}{k((j, b))})$  ;

```

---

Du fait de la structure de données  $k((i, a), (j, b))$ , l'algorithme a une complexité spatiale en  $\mathcal{O}(ed^2)$ . Il est possible de se débarrasser de ces compteurs binaires en observant que les demandes de coût sur  $w_{ij}(a, b)$  ne peuvent provenir que de la variable  $i$  ou  $j$ .  $k((i, a))$  demandes sont faites par  $i$  si  $\text{tueur}(i, a) = j$  et  $M(i, a)$  est vrai, et de même de façon symétrique pour  $(j, b)$ . Ainsi,  $k((i, a), (j, b))$  peut être calculé à la volée en temps constant à partir de  $\text{tueur}$ , de  $M$  et des compteurs unaires  $k$ . On obtient ainsi une complexité spa-

---

**Algorithme 3** : VAC - Phase 3 : Application des EPT

---

```

tant que ( $R \neq \emptyset$ ) faire
  ( $j, b) \leftarrow R.Pop()$  ;
   $i \leftarrow \text{tueur}(j, b)$  ;
  pour tous les  $a \in D_i$  t.q.  $k_j((i, a)) \neq 0$  faire
  |  $\text{Project}(w_{ij}, i, a, -\lambda \times k_j((i, a)))$  ;
  |  $k_j((i, a)) \leftarrow 0$  ;
  |  $\text{Project}(w_{ij}, j, b, \lambda \times k((j, b)))$  ;
UnaryProject( $i_0, \lambda$ ) ;

```

---

tiale en  $\mathcal{O}(ed)$ .

Une itération de l'algorithme s'effectue en  $\mathcal{O}(ed^2)$ . Ceci est vrai pour la première passe dès lors qu'un algorithme de filtrage par AC optimal est utilisé (l'instrumentation elle-même étant en  $\mathcal{O}(nd)$ ). La 2nd phase est en  $\mathcal{O}(nd^2)$  car il y a au plus  $nd$  valeurs dans  $P$  et la boucle de la ligne 4 est en  $\mathcal{O}(d)$ . En s'appuyant sur l'astuce réduisant la complexité spatiale à  $\mathcal{O}(ed)$  ci-dessus, la même complexité en  $\mathcal{O}(nd^2)$  s'applique à la dernière phase. Comme  $\lambda$  peut devenir de plus en plus petit après chaque itération, le nombre d'itérations de VAC n'a pu être borné. Pour implémenter VAC, nous avons utilisé un seuil  $\varepsilon$ . Si un nombre fixé d'itérations ne mène jamais à une augmentation de  $w_\emptyset$  supérieure à  $\varepsilon$ , alors l'établissement de VAC est arrêté de façon prématuré. On parle d'établissement de  $\text{VAC}_\varepsilon$ . Le nombre d'itérations est alors en  $\mathcal{O}(\frac{m}{\varepsilon})$ . Quand une itération n'augmente pas le minorant par plus de  $\varepsilon$ , un goulot d'étranglement (un coût qui a déterminé la valeur de  $\lambda$ ) est identifié et les coûts unaires et binaires associés à une variable impliquée dans ce goulot sont ignorés dans  $\text{Bool}(P)$  dans les itérations suivantes.

De façon à rapidement collecter des contributions importantes, on remplace  $\text{Bool}(P)$  par une version relaxée mais de plus en plus stricte notée  $\text{Bool}_\theta(P)$ . Un tuple  $t$  est interdit dans  $\text{Bool}_\theta(P)$  si et seulement si son coût dans  $P$  est plus grand que  $\theta$ . Après avoir trié la liste des coûts binaires non nuls  $w_{ij}(a, b)$  dans un nombre  $h$  de sacs (buckets), la séquence décroissante des coûts minimum de chaque sac définit une séquence de seuils  $(\theta_1, \dots, \theta_h)$ . A partir de  $\theta_1$ , des itérations de VAC sont effectuées à un seuil fixe jusqu'à absence de wipe-out. On passe alors à la valeur  $\theta_{i+1}$  suivante. Quand  $\theta_h$  est atteint, un schéma géométrique définit par  $\theta_{i+1} = \frac{\theta_i}{2}$  est utilisé et s'arrête quand  $\theta_i \leq \varepsilon$ .

## 5 Résultats expérimentaux

Dans cette section, nous présentons le résultat d'expérimentations de  $\text{VAC}_\varepsilon$  réalisées sur des problèmes

réels et générés aléatoirement en utilisant `toulbar2`<sup>1</sup>. Notre implémentation s’appuie sur une représentation des coûts en virgule fixe. Pour cela, tous les coûts initiaux du problème sont multipliés par  $\frac{1}{\varepsilon}$ , supposé entier. Afin d’exploiter le fait que le problème original résolu est à coût entier, l’algorithme de séparation-évaluation élargue dès lors que  $\frac{\lceil w_\theta \times \varepsilon \rceil}{\varepsilon} \geq ub$  où  $ub = m$  est le majorant global courant (coût de la meilleure solution connue).

Les tests sont effectués sur un Intel Xeon 3 GHz avec 16 GB de mémoire. Notre solveur inclut également une méthode de choix de variable dirigé par le dernier conflit, effectuée de l’élimination de variable à la volée et un branchement dichotomique. Lorsque  $VAC_\varepsilon$  est utilisée, nous avons fixé par défaut  $\varepsilon = \frac{1}{10000}$ . Cette valeur est constante dans tous les tests qui suivent (et semble robuste). Il est possible de maintenir  $VAC_\varepsilon$  pendant la recherche. Chaque itération de  $VAC_\varepsilon$  nécessite une reconstruction de  $Bool_\theta(P)$ , et est donc assez lourde. De ce fait, la convergence de  $VAC_\varepsilon$  est arrêté de façon prématurée pendant la recherche en utilisant un seuil  $\theta$  plus grand que durant le prétraitement. Ceci permet de n’établir VAC que lorsqu’il est capable de fournir des augmentations du minorant consécutives. Pour les problèmes aléatoires, le majorant initial utilisé est égal à 0.

**Instances générées aléatoirement** Le premier jeu d’instances est formé de Max-CSP aléatoires. Nous avons utilisé les problèmes de [5]. Ce sont les jeux “Sparse Tight”, “Dense Tight” et “Complete Tight” (ST, DT, CT, 32 variables, 10 valeurs, 50 instances par classe) sur lesquelles le filtrage par  $VAC_\varepsilon$  et OSAC fournit des minorants non triviaux. La table ci-dessous donne le temps nécessaire et la qualité moyenne du minorant (lb) obtenu après filtrage par EDAC,  $VAC_\varepsilon$  et OSAC :

prétraitement	ST		DT		CT	
	lb	temps	lb	temps	lb	temps
	EDAC	16	<.01s	18	<.01s	40
$VAC_\varepsilon$	25	.06s	28	.09s	49	.25s
OSAC	27	10.5s	32	2.1s	74	631s

Comme on l’attendait, OSAC fournit toujours les minorants les plus forts.  $VAC_\varepsilon$  fournit un minorant qui est 8% (ST) à 33% (CT) plus faible que celui d’OSAC en étant plus rapide d’un ou deux ordres de grandeur.

Pour évaluer l’efficacité de  $VAC_\varepsilon$  sur des problèmes sous-modulaires binaires, nous avons généré des problèmes aléatoires sous-modulaires permutés. Au niveau unaire, chaque valeur reçoit un coût de 0/1 avec la même probabilité. Les fonctions de coût sous-modulaires binaires peuvent être décomposées en une

somme de *generalized interval functions* [3]. Ces fonctions sont définies par un coût fixe (nous avons utilisé 1) et une valeur  $a$  (resp.  $b$ ) dans chacune des variables impliquées. Nous avons additionné ensemble  $d$  fonctions de ce type en utilisant des valeurs aléatoires pour  $a$  et  $b$  échantillonnées de façon uniforme dans les domaines pour générer chaque fonction de coût sous-modulaire binaire. les domaines des variables sont ensuite permutés pour cacher la sous-modularité.

La table ci-dessous montre que le maintien de  $VAC_\varepsilon$  permet de rapidement surpasser EDAC sur ces instances. Les problèmes considérés ont de 100 à 150 variables, 20 valeurs par variable, et de 900 à 1450 fonctions de coût. 10 instances sont considérées par classe. Le temps CPU donné pour la résolution des problèmes inclut la preuve d’optimalité (un tiret indique un temps de plus de  $> 10^4$  secondes) :

n vars	100	110	120	130	140	150
EDAC	17	85	135	433	1363	-
$VAC_\varepsilon$	0.31	0.34	0.37	0.36	0.77	1.17

**Affectation de fréquences** Le problème d’affectation de fréquence (RLFAP) [2] consiste à affecter des fréquences à un ensemble de liens de communication radio de façon à ce que les différents liens puissent opérer simultanément sans interférence notable. Certaines instances de ce problème peuvent se modéliser simplement sous forme de CSP pondéré binaire.

Nous avons d’abord comparé  $VAC_\varepsilon$  à OSAC et EDAC en prétraitement seulement. Les instances de RLFAP du CÉLAR (que nous remercions ici) sont distribuées dans leur formulation originale ou prétraitées par une combinaison d’analyse de dominance et de singleton AC pondérée. Un indice  $r$  dans le nom de l’instance identifie les instances réduites prétraitées (avec le même coût optimal que les instances originales).

La table ci-dessous montre que  $VAC_\varepsilon$  est plus rapide qu’OSAC d’un à deux ordres de grandeur et fournit un minorant proche de celui d’OSAC sur les instances ouvertes `graph11r` et `graph13r`.

		scen07 <sub>r</sub>	scen08 <sub>r</sub>	graph11 <sub>r</sub>	graph13 <sub>r</sub>
prétraitement	EDAC	10000	6	2710	8722
	$VAC_\varepsilon$	29498	35	2955	9798
cpu	OSAC	31454	48	2957	9798
	$VAC_\varepsilon$	211s	86s	3.5s	29s
	OSAC	3530s	6718s	492s	6254s

Nous avons tenté de résoudre ces instances en maintenant simultanément EDAC (utile pour guider les heuristiques car créant des coût unaires et  $VAC_\varepsilon$ ). Durant la recherche,  $VAC_\varepsilon$  a été arrêté à  $\theta = 10/\varepsilon$  pour toutes les instances. Le choix de cette valeur a une influence sensible sur les temps de calcul et mérite sans doute d’être ajustée selon les problèmes. La table

<sup>1</sup><http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro>.

ci-dessous donne les résultats sur l’instance `scen06` et sur les instances ouvertes `graph11` et `graph13` (voir [fap.zib.de/problems/CALMA](http://fap.zib.de/problems/CALMA)) pour lesquelles une preuve d’optimalité est fournie pour la première fois, aussi bien dans leur forme réduite qu’originale. Le majorant initial fourni est le coût de la meilleure solution connue. Dans les deux cas, ce majorant était effectivement un optimum. La table donne pour chaque instance le nombre de variables, le nombre total de valeurs, de fonctions de coût, le temps CPU pour EDAC seul, le nombre de nœuds développés avec VAC, le temps CPU avec VAC et le nombre total d’itérations VAC (nb. iter). Nous avons observé que le nombre de demandes de coût  $k$  lors d’une itération VAC peut être élevé, avec une valeur moyenne de 16 dans certaines résolutions des instances `graph`.

	nb. var.	nb. val	nb. f.c.	EDAC cpu	VAC nœuds	VAC cpu	nb. iter
gr11 <sub>r</sub>	232	5747	792	-	1536	18.2s	973
gr11	340	12820	1425	-	$2 \cdot 10^5$	217min.	$2.6 \cdot 10^5$
gr13 <sub>r</sub>	454	13153	2314	-	32	62s	1893
gr13	458	17588	4815	-	114	254s	9486
sc06	82	3274	327	39min.	$2 \cdot 10^6$	155min.	$3 \cdot 10^6$

**Affectation d’entrepôts** Nous avons testé le pré-traitement par  $VAC_\varepsilon$  sur des instances de problème d’affectation d’entrepôts (uncapacited warehouse location problem ou UWLP, décrit et modélisé comme CSP pondéré dans [12] et [8] respectivement). Les temps de résolution sur les instances `capmq1-5` (600 variables, 300 valeurs max. par variable et 90000 fonctions de coûts), `capa`, `capb` et `capc` (1100 variables, environ 90 valeurs par variable et 101100 fonctions de coût) sont donnés dans la table ci-dessous :

	mq1	mq2	mq3	mq4	mq5	a	b	c
EDAC	2508	3050	2953	7052	7323	6179	-	-
$VAC_\varepsilon$	2279	3312	2883	4024	8124	3243	4343	2751
$CPLEX_\varepsilon$	622	1022	415	1266	2357	3	4.5	13

Ces instances ont également été résolues en utilisant le moteur de programmation linéaire en nombres entiers CPLEX 11.0 sur une formulation directe du problème. Il faut noter que, du fait du large intervalle de variation des coûts dans ces instances, représentés par des nombres flottants dans CPLEX, la preuve d’optimalité de CPLEX n’est pas véritablement garantie, contrairement à VAC, utilisant une représentation exacte. Les résultats d’OSAC ne sont pas donnés car la génération du programme linéaire dépasse la capacité machine sur ces instances.

## 6 Travaux connexes

Comme OSAC [5], VAC cherche à atteindre une reformulation d’un problème via des transformations au niveau arc maximisant la valeur de la fonction de coût constante. OSAC identifie un *ensemble* de transformation au niveau arc qui appliquées simultanément fournissent une reformulation optimale. VAC tente seulement de construire des *séquences* de transformations qui appliquées séquentiellement augmente la valeur de la fonction de coût constante.

L’exploitation de cohérences locales classiques pour fournir des minorants de CSP pondérés ou d’instances Max-SAT n’est pas nouvelle. Dans les Max-CSP, [14] utilise des sous-problèmes arc-incohérents indépendants pour fournir un minorant. De façon similaire, [13] s’appuie sur des ensemble de clauses disjoints détectés comme incohérents par propagation unitaire (UP) pour fournir un minorant. Ces approches ne transforment pas les problèmes traités et exploitent le fait que les problèmes incohérents détectés sont indépendants, permettant de simplement additionner les coûts correspondants. Elles n’ont pas le caractère incrémental des cohérences locales.

Pour les problèmes Max-SAT, [9] exploite également les incohérences détectées par UP pour construire des séquences de transformations *entières* mais possiblement au delà du niveau arc ce qui implique la génération de clauses pondérées (fonctions de coût) d’arité plus large que 2. Notre approche reste au niveau arc en autorisant des mouvements de coût dans  $\mathbb{Q}$ . Elle peut être vue comme une généralisation non triviale (capable de traiter les cas non booléens et non binaires) de l’algorithme de construction du “roof-dual” basé sur un algorithme de flot maximum utilisé en optimisation pseudo-booléenne quadratique [1]. Elle est très proche de l’algorithme “Augmenting DAG” proposé indépendamment par [18] pour le traitement de grammaires bi-dimensionnelles, et récemment traduit dans [19].

## 7 Conclusion

Cet article montre comment la cohérence d’arc classique peut être utilisée pour identifier des séquences de transformations au niveau arc permettant d’augmenter la fonction de coût constante. En autorisant la manipulation de coûts rationnels, VAC est capable de fournir des minorants renforcés en se limitant aux opérations de niveau arc, permettant d’obtenir une incrémentalité importante dans une recherche arborescente.

Notre algorithme pour établir VAC reste très préliminaire. Une part du travail réalisé à chaque itération reste inutile et pourrait être réexploité aux itérations

suivantes (certains effacements hors du sous-ensemble minimal identifié). La dernière phase de l'algorithme effectuant implicitement une relaxation de  $\text{Bool}(P)$  à chaque itération, l'utilisation d'algorithmes d'AC dynamiques pourrait être judicieuse. L'utilisation d'heuristiques dans l'algorithme instrumenté de filtrage par AC mérite aussi d'être étudié car l'augmentation de coût obtenue à chaque itération dépend de la preuve d'incohérence de  $\text{Bool}(P)$  construite. Bien que fondamentalement différent d'un algorithme de flot maximum, VAC pourrait sans doute tirer parti des améliorations proposées dans les algorithmes de flot maximum récents. Enfin, l'algorithme présenté a été implémenté sur des fonctions de coût binaires mais il s'étend aux fonctions de coût non binaires et à d'autres structures de valuations, qui devrait permettre de traiter les problèmes Max-SAT ou de recherche d'explication de probabilité maximum dans les réseaux probabilistes (bayésiens ou markoviens). L'instrumentation de contraintes globales dures pour établir VAC constitue une autre direction de recherche intéressante.

**Remerciements** Nous voudrions remercier Tomas Werner pour les discussions autour de l'algorithme "Augmenting DAG". Ce travail a été en partie financé par l'Agence Nationale de la Recherche (projet STAL-DECOPT).

## Références

- [1] E. Boros and P. Hammer. Pseudo-Boolean Optimization. *Discrete Appl. Math.*, 123 :155–225, 2002.
- [2] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4 :79–89, 1999.
- [3] DA. Cohen, MC. Cooper, PG. Jeavons, and AA. Krokhin. A Maximal Tractable Class of Soft Constraints. *Journal of Artificial Intelligence Research*, 22 :1–22, 2004.
- [4] DA. Cohen, MC. Cooper, PG. Jeavons, and AA. Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11) :983 – 1016, August 2006.
- [5] MC. Cooper, S. de Givry, and T. Schiex. Optimal soft arc consistency. In *Proc. of IJCAI-07*, pages 68–73, Hyderabad, India, January 2007.
- [6] MC. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154 :199–227, 2004.
- [7] MC. Cooper. Minimization of locally-defined submodular functions by Optimal Soft Arc Consistency. *Constraints*, 13(4), 2008.
- [8] S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted csps. In *Proc. of IJCAI-05*, pages 84–89, Edinburgh, Scotland, 2005.
- [9] F. Heras, J. Larrosa, and A. Oliveras. MiniMax-Sat : A New Weighted Max-SAT Solver. In *Proc. of SAT'2007*, number 4501 in LNCS, pages 41–55, Lisbon, Portugal, May 2007.
- [10] PG. Jeavons and MC. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2) :327–339, December 1995.
- [11] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *Proc. of the 17<sup>th</sup> IJCAI*, pages 322–327, Seattle, WA, 2001.
- [12] J. Kratica, D. Tomic, V. Filipovic, and I. Ljubic. Solving the Simple Plant Location Problems by Genetic Algorithm. *RAIRO Operations Research*, 35 :127–142, 2001.
- [13] CM. Li, F. Manyà, and J. Planes. Exploiting Unit Propagation to Compute Lower Bounds in Branch and Bound Max-SAT Solvers. In *Proc of CP-05*, number 3709 in LNCS, pages 403–414, Sitges, Spain, 2005.
- [14] JC. Régim, T. Petit, C. Bessière, and JF. Puget. New Lower Bounds of Constraint Violations for Over-Constrained Problems. In *Proc. of CP-01*, number 2239 in LNCS, pages 332–345, Paphos, Cyprus, December 2001.
- [15] T. Schiex. Arc consistency for soft constraints. In *Proc. of CP-00*, volume 1894 of LNCS, pages 411–424, Singapore, 2000.
- [16] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems : hard and easy problems. In *Proc. of IJCAI-95*, pages 631–637, Montréal, Canada, August 1995.
- [17] D. Schlesinger. Exact Solution of Permuted Submodular MinSum Problems. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, number 4679/2007 in LNCS, pages 28–38, August 2007.
- [18] MI. Schlesinger. Sintaksicheskiy analiz dvumernykh zritelnykh signalovv usloviyakh pomekh (Syntactic analysis of two-dimensional visual signals in noisy conditions). *Kibernetika*, 4 :113–130, 1976.
- [19] T. Werner. A Linear Programming Approach to Max-sum Problem : A Review. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 29(7) :1165–1179, July 2007.

- [20] 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*, pages 394–398, Dijon, France, April 2006.