CoQuiAAS : Applications de la programmation par contraintes à l'argumentation abstraite

Jean-Marie Lagniez Emmanuel Lonca Jean-Guy Mailly
CRIL – Université d'Artois, CNRS
Lens, France
{lagniez,lonca,mailly}@cril.univ-artois.fr

Résumé

L'argumentation est aujourd'hui un des mots-clés prépondérants en ce qui concerne l'intelligence artificielle. Elle a notamment sa place dans des thématiques comme les systèmes multi-agent, où elle permet la mise en place de protocoles de dialogue (à base de persuasion, de négociation,...) ou l'analyse de discussions en ligne ; mais aussi dans le cas où un unique agent doit raisonner en présence d'informations conflictuelles (inférence en présence d'incohérence, mesures d'incohérence). Ce cadre très riche offre la possibilité de raisonner suivant un nombre important de sémantiques et propose deux politiques d'inférence pour chaque sémantique.

D'autre part, les travaux réalisés sur SAT ces dernières années, et de façon générale dans le domaine de la programmation par contraintes, ont permis d'obtenir des approches efficaces en pratique pour traiter des problèmes dont la complexité théorique est élevée.

Les besoins d'applications efficaces pour effectuer les tâches de raisonnement usuelles à partir d'un système d'argumentation, ainsi que la puissance des techniques modernes de programmation par contraintes, nous ont amené à étudier l'encodage des sémantiques usuelles en argumentation abstraite dans des formalismes logiques, afin de pouvoir utiliser diverses approches de la programmation par contraintes pour développer une bibliothèque logicielle de raisonnement argumentatif. La bibliothèque que nous proposons présente l'avantage d'être générique et aisément adaptable. Après une présentation de la conception de notre logiciel, nous étudions expérimentalement notre approche sur un ensemble de sémantiques et de tâches d'inférence usuelles.

1 Introduction

Un système d'argumentation abstrait [15] est un graphe orienté dont les nœuds représentent des entités abstraites appelées *arguments* et dont les arêtes représentent des *attaques* entre ces arguments. Ce cadre simple et élégant est utilisé aussi bien dans des traitements concernant un seul agent que dans des scénarios multi-agent. En ce qui concerne le cas d'un unique agent, celui-ci peut par exemple utiliser un système d'argumentation pour raisonner à partir d'informations contradictoires, ce qui peut entre autre lui permettre de construire un système d'argumentation à partir d'une base de connaissances incohérente pour en tirer des conséquences non triviales [8]). Dans un cadre multi-agent, on peut utiliser l'argumentation pour modéliser des dialogues entre plusieurs agents [3] ou pour analyser une discussion en ligne entre utilisateurs de réseaux sociaux [24]. Le sens d'un tel graphe est déterminé par une *sémantique d'acceptabilité*, qui indique quelles propriétés un ensemble d'arguments doit satisfaire pour que cet ensemble soit considéré comme une « solution » acceptable du problème ; on appelle alors cet ensemble une *extension*.

À l'heure actuelle, une des tendances fortes dans la communauté de l'argumentation est de développer des approches logicielles pour calculer diverses tâches d'inférence à partir d'un système d'argumentation pour les sémantiques usuelles (voir `http://argumentationcompetition.org/2015` pour plus de détails). Pour une sémantique donnée, les requêtes usuelles concernent majoritairement le calcul d'une extension, de toutes les extensions, ou bien encore l'acceptation crédule (respectivement sceptique) d'un argument particulier, c'est-à-dire son appartenance à au moins une (respectivement chaque) extension du système d'argumentation. Il est connu que pour la plupart des couples composés d'une sémantique et d'une requête usuelles, la complexité théorique des problèmes est élevée [14, 17]. Ces tâches nécessitent de ce fait

de disposer d'outils efficaces en pratique pour pouvoir être calculées en temps raisonnable. Pour parvenir à calculer efficacement les extensions d'un système d'argumentation, ainsi que pour parvenir à raisonner sur l'acceptation crédule ou sceptique d'un argument en temps raisonnable, nous proposons dans cet article d'utiliser diverses techniques liées à la programmation par contraintes, domaine dont les approches proposées apportent des solutions efficaces en terme de raisonnement logique pour les problèmes combinatoires. Nous nous intéressons tout particulièrement dans cet article à la logique propositionnelle et aux formalismes dérivés de cette dernière. Plus précisément, nous proposons des encodages sous forme normale conjonctive pour des problèmes de décision du premier niveau de la hiérarchie polynomiale, mais aussi des encodages dans le formalisme *Partial Max-SAT*; ces encodages nous permettent de répondre à des requêtes concernant quatre sémantiques et quatre requêtes usuelles. Nous tirons ensuite parti de ces encodages pour résoudre les problèmes de notre étude en utilisant des logiciels et approches de l'état de l'art ayant démontré leur efficacité pratique.

Nous avons encodé ces différentes approches pour le raisonnement à partir d'un système d'argumentation dans une bibliothèque logicielle appelée CoQuiAAS. Le but de notre bibliothèque est non seulement de fournir des algorithmes de base permettant de gérer les principales requêtes pour les principales sémantiques, mais aussi de fournir un framework évolutif dans lequel l'ajout de nouveaux paramètres (requête, sémantique) ou la réalisation de nouveaux algorithmes de résolution pour des problèmes déjà gérés est aisée.

Dans cet article, nous présentons en premier lieu les notions de base concernant l'argumentation abstraite et les problèmes vers lesquels nous réduisons nos requêtes, à savoir le problème SAT (recherche d'un modèle dans une formule sous forme normale conjonctive) et le problème de la recherche de sous-ensembles maximaux cohérents (MSS) dans des instances dites *Partial Max-SAT*. Nous détaillons à la suite de cette présentation les encodages que nous avons employés afin de réduire les problèmes d'argumentation considérés aux problèmes de vérification de cohérence ou de recherche de sous-ensemble maximal cohérent. Nous présentons ensuite dans la section 4 la façon dont nous avons conçu notre bibliothèque, CoQuiAAS. Enfin, nous donnons des résultats expérimentaux concernant les encodages que nous proposons en section 5, et nous comparons d'un point de vue conception notre bibliothèque aux logiciels existants capables de calculer des requêtes faisant parti de notre étude.

2 Préliminaires formels

2.1 Logique propositionnelle

L'alphabet de la logique propositionnelle est composé d'un ensemble de variables booléennes et d'un ensemble de trois opérateurs usuels : l'opérateur de négation $\neg$ (unaire), l'opérateur de conjonction $\wedge$ (binaire) et l'opérateur de disjonction $\vee$ (binaire), chacun de ces opérateurs connectant des formules entre elles (les variables propositionnelles étant elle-même des formules). Dans la mesure où une formule propositionnelle est construite de manière inductive (puisque les connecteurs s'appliquent sur des formules), elle peut de ce fait être vue comme un arbre. Étant donnée une affectation de l'ensemble des variables propositionnelles (appelée interprétation), une formule propositionnelle est évaluée à *vrai* si et seulement si le nœud racine de la formule est évalué à *vrai*. Pour connaitre la valeur de ce nœud, il suffit de procéder au calcul de chacun des nœud dans l'ordre topologique inverse ; en effet la valeur des feuilles est connue (puisque les feuilles sont les nœuds correspondant aux variables), et la valeur des nœuds internes peut être déterminée en fonction de la sémantique attachée aux connecteurs qui leurs sont associés : un nœud négation ($\neg$) a pour valeur *vrai* si et seulement si son fils a la valeur *faux*, un nœud $\wedge$ (respectivement $\vee$) a la valeur *vrai* si et seulement si ses deux (respectivement un de ses deux fils) fils ont (respectivement a) pour valeur *vrai*. Lorsqu'une interprétation donne à une formule la valeur *vrai*, on dit que c'est un *modèle* de la formule. Par convention, on représente une interprétation par l'ensemble des variables affectées à *vrai* par cette interprétation. Usuellement, on définit en sus des trois opérateurs présentés les connecteurs d'implication ($\Rightarrow$, tel que $a \Rightarrow b \equiv \neg a \vee b$) et d'équivalence ($\Leftrightarrow$, tel que $a \Leftrightarrow b \equiv (a \Rightarrow b) \wedge (b \Rightarrow a)$). Lorsqu'une formule propositionnelle Φ est équivalente à une conjonction $\varphi_1 \wedge \cdots \wedge \varphi_n$, on peut représenter Φ comme un ensemble $\{\varphi_1, \ldots, \varphi_n\}$.

Dans notre étude, nous traitons d'encodages en formules NNF (*Negation Normal Form*), c'est-à-dire des formules propositionnelles dans lesquelles les négations ne portent que sur des variables (nœuds terminaux). Cependant, les prouveurs SAT n'étant capables que de gérer des formules CNF (*Conjunctive Normal Form*), une étape de traduction de formule NNF vers CNF est nécessaire entre les encodages présents dans l'état de l'art et ceux que nous utilisons de manière effective ; ceci n'influe cependant pas la généralité de nos approches dans la mesure où, pour toute formule propositionnelle, on peut déterminer en temps polynomial une formule CNF équivalente.

Les formules CNF correspondent aux conjonctions

de disjonction de littéraux (un littéral est une variable propositionnelle potentiellement niée), c'est-à-dire les formules écrites sous la forme $\wedge_{i=1}^{n}(\vee_{j=1}^{n_i} l_{i,j})$. Ces formules sont intéressantes dans la mesure où une disjonction de littéraux (appelée *clause*) permet de représenter aisément une contrainte ; une formule CNF est ainsi en fait un ensemble de contraintes, et un modèle d'une formule CNF est alors une affectation des variables telle que toutes les contraintes d'un problème soient satisfaites.

Bien que ce formalisme soit adapté à la représentation d'un problème, la recherche d'un modèle pour une formule CNF est théoriquement complexe (NP-difficile [13]). Cependant, des logiciels (appelés *prouveurs SAT*, ou encore *solveurs SAT*) sont dédiés à la résolution de tels problèmes et permettent d'en résoudre de plus en plus imposant [9]. Il existe néanmoins des problèmes pour lesquels il n'existe aucun modèle, c'est-à-dire aucune affectation complète des variables telle que l'ensemble des contraintes soit satisfait. On peut dans ce cas chercher à déterminer une interprétation qui maximise le nombre de contraintes résolues : on nomme ce problème *Max-SAT* [9]. On peut généraliser ce problème en donnant un poids à chacune des contraintes (on cherche alors à maximiser la somme des poids des contraintes satisfaites) ; il s'agit alors du problème *Weighted Max-SAT*. Si des contraintes ont un poids infini (elles doivent impérativement être satisfaites), on passe alors dans le cadre des problèmes *Partial Max-SAT* et *Weighted Partial Max-SAT*.

Déterminer une solution à un problème Max-SAT permet donc de déterminer un ensemble de contraintes de la formule initiale qui est cohérent, tel que l'ajout de n'importe quelle autre contrainte issue du problème de départ rend ce nouvel ensemble incohérent [25] ; un sous-ensemble de contraintes d'une formule ayant cette propriété est un *sous-ensemble maximal cohérent* (MSS) [26]. Étant donné l'ensemble de contraintes φ d'un problème et ψ un ensemble de contraintes formant un MSS de φ, on dit que $\overline{\psi} = \varphi \setminus \psi$ est un *coMSS* (ou *MCS*) de la formule [25]. Notons toutefois que les solutions optimales pour le problème Max-SAT ne forment qu'un sous-ensemble des MSS d'une formule.

Il est intéressant de noter qu'en ce qui concerne les algorithmes développés en programmation par contraintes pour la résolution de problème Max-SAT ou pour la recherche de MSS, l'approche utilisée consiste généralement à utiliser comme boite noire un prouveur SAT classique basé sur l'interface incrémentale de Minisat [4, 20, 23] de manière successive sur différents problèmes de test de cohérence. En ce qui concerne le calcul de MSS par le biais d'un tel prouveur, on peut par exemple citer l'algorithme BLS [5] ou plus récemment l'algorithme CMP [22] ; notons toutefois que certains logiciels dévolus au calcul de MSS utilisent à cette fin un prouveur Max-SAT comme boite noire [25].

2.2 Argumentation abstraite

Plusieurs modèles ont été proposés pour formaliser l'argumentation. Nous travaillons avec l'un des plus populaires, le cadre proposé par Dung [15].

Définition 1. *Un* système d'argumentation *est un graphe orienté $F = \langle A, R\rangle$ où A désigne un ensemble fini d'objets appelés des* arguments *et $R \subseteq A \times A$ une relation appelée* relation d'attaque.

La signification intuitive de la relation d'attaque est celle qui est présente lorsque l'on débat avec quelqu'un. Si un premier argument a_1 est avancé, sans opposition, rien *a priori* n'empêche de considérer que a_1 est vrai. Par contre, si quelqu'un avance un argument a_2 (*a priori* acceptable) qui attaque a_1, alors a_1 ne peut pas être accepté, à moins qu'il ne soit ensuite défendu. Cette notion intuitive de défense peut être formalisée :

Définition 2. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_1, a_2, a_3 \in A$ trois arguments.*

- *L'argument a_1 est* attaqué *par l'argument a_2 dans F si et seulement si $(a_2, a_1) \in R$.*
- *On dit alors que l'argument a_3* défend *a_1 contre a_2 dans F si et seulement si $(a_3, a_2) \in R$.*

On généralise ces notions à l'attaque et la défense d'un argument par un ensemble d'argument $E \subseteq A$.

- *L'argument a_1 est* attaqué *par l'ensemble d'arguments E dans F si et seulement si $\exists a_i \in E$ tel que $(a_i, a_1) \in R$.*
- *L'ensemble d'arguments $E \subseteq A$* défend *a_1 contre a_2 dans F si et seulement si E attaque a_2.*

Par exemple, dans le système d'argumentation décrit en Figure 1, l'argument a_1 attaque l'argument a_2, et a_2 se défend lui-même contre cette attaque.

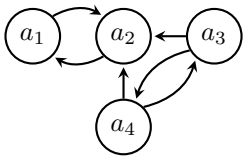

FIGURE 1 – Un exemple de système d'argumentation

Lorsque l'on raisonne avec un système d'argumentation, la première tâche est habituellement de déterminer quels ensembles d'arguments peuvent être acceptés conjointement. Différentes propriétés peuvent être exigées d'un ensemble d'arguments pour qu'il soit

considéré comme une « solution » raisonnable du système d'argumentation. Parmi ces propriétés, deux en particulier sont exigées par la plupart des sémantiques usuelles :

— l'*absence de conflit* : $E \subseteq A$ est sans conflit dans F si et seulement si $\nexists a_i, a_j \in E$ tels que $(a_i, a_j) \in R$;

— l'*admissibilité* : un ensemble sans conflit $E \subseteq A$ est admissible si et seulement si $\forall a_i \in E$, E défend a_i contre tous ses attaquants.

L'admissibilité et l'absence de conflit sont à la base des quatre sémantiques usuelles de Dung :

Définition 3. *Soit $F = \langle A, R \rangle$ un système d'argumentation.*

— *Un ensemble sans conflit $E \subseteq A$ est une* extension complète *de F si et seulement si E contient tous les arguments que défend E.*

— *Un ensemble $E \subseteq A$ est une* extension préférée *de F si et seulement si E est un élément maximal selon $\subseteq$ parmi les extensions complètes de F.*

— *Un ensemble sans conflit $E \subseteq A$ est une* extension stable *de F si et seulement si E attaque tous les arguments qui n'appartiennent pas à E.*

— *Un ensemble $E \subseteq A$ est une* extension de base *de F si et seulement si E est un élément minimal selon $\subseteq$ parmi les extensions complètes de F.*

Étant donnée une sémantique σ, on note $Ext_\sigma(F)$ les σ-extensions de F. Nous notons les sémantiques ci-dessus, respectivement, CO, PR, ST et GR.

Nous illustrons en Section 3 ces différentes sémantiques sur le système d'argumentation présenté en Figure 1.

Dung a de plus montré que pour tout système d'argumentation F,

— F admet une et une seule extension de base ;

— F admet au moins une extension préférée ;

— tout extension stable de F est une extension préférée de F ;

— F peut n'admettre aucune extension stable.

Étant donnée une sémantique σ, plusieurs problèmes de décision peuvent être considérés. En premier lieu, il peut être intéressant de déterminer le statut d'acceptation, crédule ou sceptique, d'un argument a. On définit le statut d'un argument a par :

— F accepte a sceptiquement pour la sémantique σ si et seulement si $\forall \varepsilon \in Ext_\sigma(F), a \in \varepsilon$;

— F accepte a crédulement pour la sémantique σ si et seulement si $\exists \varepsilon \in Ext_\sigma(F)$ tel que $a \in \varepsilon$.

De manière évidente, ces deux statuts coïncident pour la sémantique de base, dans la mesure où celle-ci admet une unique extension. Nous notons le problème d'acceptation crédule (respectivement sceptique) DC (respectivement DS).
Un autre problème de décision intéressant est $Exists$, qui désigne le problème consistant à vérifier s'il existe une extension non vide pour une sémantique donnée.

Le complexité pour chacun de ces problèmes de décision est résumée dans la Table 1 (dont les résultats sont issus d'autres publications [14, 17]). $\mathsf{C}-c$ signifie que le problème considéré est complet pour la classe de complexité C.

Sémantique	GR	ST	PR	CO
DC	P	NP$-c$	NP$-c$	NP$-c$
DS	P	coNP$-c$	$\Pi_2^P - c$	P$-c$
$Exist$	P	NP$-c$	NP$-c$	NP$-c$

TABLE 1 – Complexité des problèmes d'inférence pour les sémantiques usuelles

De toute évidence, la complexité du problème $Exist$ est une borne inférieure pour la complexité du calcul d'une extension, tandis que la complexité de DS est une borne inférieure pour la complexité de l'énumération des extensions.

3 Encodages logiques pour l'argumentation abstraite

Il est déjà connu que les sémantiques usuelles des systèmes d'argumentation peuvent être encodées en logique propositionnelle [7]. Nous tirons parti des encodages proposés par Besnard et Doutre pour proposer des approches permettant de calculer les extensions, mais aussi de déterminer si un argument est sceptiquement ou crédulement accepté par un système d'argumentation. Nos encodages sont basés sur un langage propositionnel $\mathcal{L}_A$, défini avec les connecteurs usuels sur l'ensemble de variables propositionnelles $V_A = \{x_{a_i} \mid a_i \in A\}$, x_{a_i} signifiant que l'argument a_i est accepté par le système d'argumentation considéré. Pour raison de lisibilité, nous subsituerons a_i à x_{a_i}.

Rappelons tout d'abord l'encodage de la sémantique stable défini dans [7].

Proposition 1. *Soit $F = \langle A, R \rangle$ un système d'argumentation. $E \subseteq A$ est une extension stable de F si et seulement si E est un modèle de la formule ci-dessous.*

$$\Phi_{st}^F = \bigwedge_{a_i \in A} [a_i \Leftrightarrow (\bigwedge_{a_j \in A | (a_j, a_i) \in R} \neg a_j)]$$

Ainsi, nous pouvons proposer des méthodes très simples pour déterminer les extensions et les statuts d'acceptation des arguments.

Proposition 2. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul d'une extension stable de F revient au calcul d'un modèle de Φ^F_{st}.*
- *L'énumération des extensions stables de F revient à l'énumération des modèles de Φ^F_{st}.*
- *Déterminer si a_i est crédulement accepté par F pour la sémantique stable revient à déterminer si $\Phi^F_{st} \wedge a_i$ est cohérente.*
- *Déterminer si a_i est sceptiquement accepté par F pour la sémantique stable revient à déterminer si $\Phi^F_{st} \wedge \neg a_i$ est incohérente.*

Exemple 1. *Lorsque l'on instancie Φ^F_{st} avec le système d'argumentation présenté en Figure 1, on obtient la formule*

$$(a_1 \Leftrightarrow \neg a_2), (a_2 \Leftrightarrow \neg a_1 \wedge \neg a_3 \wedge \neg a_4), (a_3 \Leftrightarrow \neg a_4), (a_4 \Leftrightarrow \neg a_3)$$

dont les modèles sont $\{a_1, a_3\}$ et $\{a_1, a_4\}$. Les deux extensions stables de F sont donc $Ext_{st}(F) = \{\{a_1, a_3\}, \{a_1, a_4\}\}$.

Comme nous l'avons noté dans la section précédente, les prouveurs SAT ne sont capables que de gérer des formules CNF. En ce qui concerne Φ^F_{st}, l'encodage en formule CNF que nous avons utilisé est le suivant.

$$\Psi^F_{st} = \bigwedge_{a_i \in A} (a_i \vee \bigvee_{a_j \in A|(a_j,a_i)\in R} a_j), \bigwedge_{a_i \in A} [\bigwedge_{a_j \in A|(a_j,a_i)\in R} (\neg a_i \vee \neg a_j)]$$

De façon similaire, Besnard et Doutre ont proposé un encodage en NNF de la sémantique complète.

Proposition 3. *Soit $F = \langle A, R\rangle$ un système d'argumentation. $E \subseteq A$ est une extension complète de F si et seulement si E est un modèle de la formule ci-dessous.*

$$\Phi^F_{co} = \bigwedge_{a_i \in A} [a_i \Rightarrow (\bigwedge_{a_j \in A|(a_j,a_i)\in R} \neg a_j) \wedge (a_i \Leftrightarrow (\bigwedge_{a_j \in A|(a_j,a_i)\in R} \bigvee_{a_k \in A|(a_k,a_j)\in R} a_k))]$$

De la même façon que pour la sémantique stable, nous avons dû traduire cette formule NNF en formule CNF afin de pouvoir utiliser des prouveurs SAT pour nos calculs. En revanche, contrairement au cas précédent, nous avons ajouté des variables auxiliaires P_a définies comme équivalentes à la disjonction des attaquants de l'argument a. Ces variables additionnelles nous permettent d'écrire une formule CNF Ψ^F_{co}, pour laquelle il existe une bijection entre les modèles de Φ^F_{co} et de Ψ^F_{co}, de manière plus élégante qu'en traduisant de manière naïve de NNF vers CNF.

$$\Psi^F_{co} = \bigwedge_{a_i \in A} (\neg a_i \vee \neg P_{a_i}), \bigwedge_{a_i \in A} (a_i \vee \bigvee_{a_j \in A|(a_j,a_i)\in R} \neg P_{a_j}), \bigwedge_{a_i \in A} (\bigwedge_{a_j \in A|(a_j,a_i)\in R} (\neg a_i \vee P_{a_j})), \bigwedge_{a_i \in A} (\neg P_{a_i} \vee \bigvee_{a_j \in A|(a_j,a_i)\in R} a_j), \bigwedge_{a_i \in A} [\bigwedge_{a_j \in A|(a_j,a_i)\in R} (P_{a_i} \vee \neg a_j)]$$

Proposition 4. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul d'une extension complète de F revient au calcul d'un modèle de Φ^F_{co}.*
- *L'énumération des extensions complètes de F revient à l'énumération des modèles de Φ^F_{co}.*
- *Déterminer si a_i est crédulement accepté par F pour la sémantique complète revient à déterminer si $\Phi^F_{co} \wedge a_i$ est cohérente.*
- *Déterminer si a_i est sceptiquement accepté par F pour la sémantique complète revient à déterminer si $\Phi^F_{co} \wedge \neg a_i$ est incohérente.*

Exemple 2. *Lorsque l'on instancie Φ^F_{co} avec le système d'argumentation présenté en Figure 1, on obtient la formule*

$$(a_1 \Rightarrow \neg a_2) \wedge (a_1 \Leftrightarrow a_1 \vee a_3 \vee a_4),$$

$$(a_2 \Rightarrow \neg a_1 \wedge \neg a_3 \wedge \neg a_4) \wedge (a_2 \Leftrightarrow a_2 \wedge a_4 \wedge a_3),$$

$$(a_3 \Rightarrow \neg a_4) \wedge (a_3 \Leftrightarrow a_3),$$

$$(a_4 \Rightarrow \neg a_3) \wedge (a_4 \Leftrightarrow a_4)$$

dont les modèles sont ceux de Φ^F_{st}, auxquels on ajoute $\{a_1\}$ et $\emptyset$. Les extensions complètes de F sont donc $Ext_{co}(F) = \{\{a_1, a_3\}, \{a_1, a_4\}, \{a_1\}, \emptyset\}$.

Les notions de minimalité et maximalité pour $\subseteq$ ne sont pas évidentes à mettre en œuvre directement dans un encodage propositionnel. On définit donc simplement une extension de base (respectivement préférée) comme un modèle minimal (respectivement maximal) pour $\subseteq$ de Φ^F_{co}. On remarque toutefois que l'extension de base d'un système d'argumentation peut être calculée en appliquant la propagation unitaire sur Φ^F_{co}.

Proposition 5. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul de l'extension de base de F revient au calcul des littéraux propagés dans Φ^F_{co}.*
- *Déterminer si a_i est accepté par F pour la sémantique de base revient à déterminer si a_i est propagé dans Φ^F_{co}.*

Exemple 3. *L'application de la propagation unitaire dans Φ^F_{co} définie à l'Exemple 2 donne un ensemble vide : la sémantique de base appliquée à F donne $Ext_{gr}(F) = \{\emptyset\}$.*

Le calcul des extensions préférées de F requiert un encodage légèrement différent. On remarque qu'un modèle maximal de Φ^F_{co} revient au calcul d'un MSS de la formule Φ^F_{pr} définie comme suit :

Proposition 6. *Soit $F = \langle A, R\rangle$ un système d'argumentation. $E \subseteq A$ est une extension préférée de F si et seulement si E est un MSS de la formule pondérée*

$$\Phi^F_{pr} = \{(\Phi^F_{co}, +\infty), (a_1, 1), \cdots, (a_n, 1)\}$$

Nous remarquons tout de même que les requêtes liées à la sémantique préférée ne nécessitent pas toutes d'utiliser l'extraction de MSS d'une instance Partial Max-SAT. En effet, vérifier si un argument est crédulement accepté est NP-complet pour la sémantique préférée. Cela revient en fait exactement à vérifier s'il est crédulement accepté pour la sémantique complète.

Proposition 7. *Soient $F = \langle A, R\rangle$ un système d'argumentation et $a_i \in A$ un argument.*

- *Le calcul d'une extension préférée de F revient au calcul d'un MSS de Φ^F_{pr}.*
- *L'énumération des extensions préférées de F revient à l'énumération des MSS de Φ^F_{pr}.*
- *Déterminer si a_i est crédulement accepté par F pour la sémantique préférée revient à déterminer si $\Phi^F_{co} \wedge a_i$ est cohérente.*
- *Déterminer si a_i est sceptiquement accepté par F pour la sémantique préférée revient à énumérer les MSS de Φ^F_{pr} et vérifier si chacun d'entre eux contient a_i.*

Exemple 4. *Reprenant l'exemple du système d'argumentaiton F donné en Figure 1, Φ^F_{pr} est la formule pondérée*

$$\{(\Phi^F_{co}, +\infty), (a_1, 1), (a_2, 1), (a_3, 1), (a_4, 1)\}$$

dont les MSS sont $\{a_1, a_3\}$ et $\{a_1, a_4\}$, qui sont donc les extensions préférées de F.

4 Conception de la bibliothèque CoQuiAAS

Nous avons fait le choix du langage C++ pour l'implémentation de CoQuiAAS pour ses avantages en terme de paradigme de programmation et d'efficacité. Tout d'abord, l'utilisation du paradigme objet de ce langage nous permet d'envisager une conception élégante pour notre bibilothèque (voir Figure 3), permettant notamment de maintenir et faire évoluer aisément le logiciel. De plus, le langage C++ garantit de bonnes performances du point de vue du temps de calcul, contrairement à d'autres langages orientés objet. Enfin, cela facilite l'intégration de coMSSExtractor, qui est l'outil sous-jacent pour la résolution des problèmes.

Le logiciel coMSSExtractor [22] a été développé comme une surcouche du solveur SAT Minisat [19]. L'API de coMSSExtractor permet à notre logiciel CoQuiAAS d'appeler ses fonctionnalités de solveur SAT et ses fonctionnalités d'extracteur de MSS/coMSS.

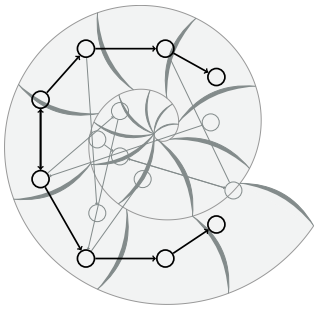

FIGURE 2 – Le logo de CoQuiAAS

Le cœur de notre application est l'interface **Solver**, qui contient les méthodes nécessaires à la résolution des problèmes étudiés. `initProblem` doit être écrite de manière à ce qu'une classe réalisant l'interface **Solver** intègre les données d'entrée, à savoir la formule logique correspondant au système d'argumentation et au problème traité. La méthode `computeProblem` effectue alors les calculs correspondant à la résolution du problème, et `displaySolution` permet d'afficher le résultat au format demandé par la *First International Competition on Computational Models of Argumentation.*

La classe abstraite *SATBasedSolver* (respectivement *CoMSSBasedSolver*) réunit les fonctionnalités et initialisations communes à tous les solveurs basés sur un prouveur SAT (respectivement un extracteur de coMSS), comme par exemple la méthode `hasAModel` qui retourne un booléen indiquant si l'instance SAT produite à partir du problème d'argumentation traité est cohérente ou non. Parmi les classes filles de *SATBasedSolver*, on trouve *DefaultSATBasedSolver* et ses classes filles, qui sont dédiées à l'utilisation de l'API de coMSSExtractor pour employer ses fonctionnalités de solveur SAT héritées de Minisat afin de résoudre les problèmes traités. Si l'utilisateur souhaite faire appel au solveur SAT de son choix au lieu de coMSSExtractor, une option de la ligne de commande

indique à la **SolverFactory** de retourner une instance de la classe *ExternalSATBasedSolver*, elle-aussi classe fille de *SATBasedSolver*, qui est initialisée avec une commande à exécuter afin d'employer tout logiciel externe capable de lire une formule CNF au format DIMACS et de retourner une solution en utilisant le formalisme imposé dans les compétitions de prouveurs SAT. Cette classe permet d'exécuter la commande fournie à CoQuiAAS pour effectuer les calculs liés à la résolution du problème, et peut par exemple permettre de comparer l'efficacité relative de plusieurs solveurs SAT différents sur les instances d'argumentation, dès lors que la sémantique considérée l'autorise. La même conception se retrouve du côté de la classe *CoMSSBasedSolver*, qui peut être instanciée *via* le solveur par défaut *DefaultCoMSSBasedSolver*, qui fait appel aux fonctions de l'API de CoMSSExtractor, ou *via* la classe *ExternalCoMSSBasedSolver* pour utiliser un logiciel tiers dont les entrées et sorties correspondent à celles de coMSSExtractor, pour les couples requête/sémantique gérés.

Notre conception est suffisamment souple pour permettre une évolution aisée de la bibliothèque CoQuiAAS. Il est par exemple simple de créer un prouveur basé sur l'API d'un prouveur SAT autre que coMSSExtractor : il suffit de créer une nouvelle classe **MySolver** qui hérite de *SATBasedSolver* (et donc de l'interface mère de tous les prouveurs, **Solver**), puis d'implémenter les méthodes abstraites de cette classe, à savoir les fonctions `initProblem`, `hasAModel`, `getModel` et `addBlockingClause`. Il est aussi possible, par exemple, d'hériter directement de la classe **Solver** et d'écrire ses trois méthodes `initProblem`, `computeProblem` et `displaySolution` pour créer un prouveur quelconque. Par exemple, si l'on souhaite développer des solveurs de problèmes d'argumentation utilisant le cadre des CSP, en se basant sur des encodages comme ceux présentés dans [2], il suffit d'ajouter une nouvelle classe *CSPBasedSolver* qui implémente l'interface **Solver**, et de reproduire le processus qui a mené à la conception des solveurs basés sur SAT, en se basant cette fois-ci sur l'API d'un solveur CSP (ou sur un solveur CSP externe).

Une fois le prouveur rédigé, il suffit ensuite de relier une option de la ligne de commande à l'utilisation de ce prouveur, en mettant à jour pour cela la méthode `getSolverInstance` de la **SolverFactory**, qui dispose parmi ses paramètres de l'ensemble des arguments de la ligne de commande de CoQuiAAS, via la map `opt`. On peut par exemple relier le paramètre `-solver MySolver` à l'utilisation de la classe **MySolver** dédiée au nouveau solveur en ajoutant le simple code suivant.

```
if(!opt["-solver"].compare("MySolver"))
    return new MySolver(...);
```

De la façon dont l'interface **Solver** est conçue, il est supposé qu'un solveur soit consacré à un unique problème pour une unique sémantique. Il est ainsi possible d'implémenter une classe dédiée à une sémantique et un problème particuliers, utilisant un algorithme adapté à ceux-ci. Par exemple, [10] décrit une procédure qui permet de déterminer si un argument donné est dans l'extension de base d'un système d'argumentation. On peut envisager d'implémenter une classe **GroundedDiscussion** qui implémente elle-même l'interface **Solver** afin de résoudre le problème d'inférence sceptique sous la sémantique de base.

Ce comportement de CoQuiAAS n'empêche cependant pas l'implémentation de prouveurs capables de résoudre plusieurs requêtes d'une même sémantique, à partir du moment où la **SolverFactory** redirige les différents problèmes à traiter vers la même classe. Ainsi, étant donnée la possibilité d'utiliser, pour une sémantique donnée, différentes approches toutes basées sur SAT (ou les coMSS) quel que soit le problème traité, nous avons simplifié la conception de nos solveurs en utilisant une classe par sémantique. Par exemple, la méthode `computeProblem` est implémentée de la façon décrite dans l'Algorithme 1 dans la classe **CompleteSemanticSolver**.

Algorithme 1 : `computeProblem`

Données : Un système d'argumentation F, un problème P, un argument a_i
suivant P **faire**
  cas où *Calcul d'une extension*
    computeOneExtension(F)
  cas où *Énumération des extensions*
    computeAllExtensions(F)
  cas où *Acceptation crédule*
    checkCredulousAcceptance(F,a_i)
  cas où *Acceptation sceptique*
    checkSkepticalAcceptance(F,a_i)

Notons que la valeur du paramètre a_i est ignorée lorsque le problème considéré est le calcul d'une extension ou l'énumération des extensions.

5 Expérimentations

Nous avons mené nos expérimentations sur les jeux d'instances fournis par les organisateurs de la *First International Competition on Computational Models of Argumentation* afin d'éprouver les solveurs en amont de la dite compétition. Le premier jeu d'instance se

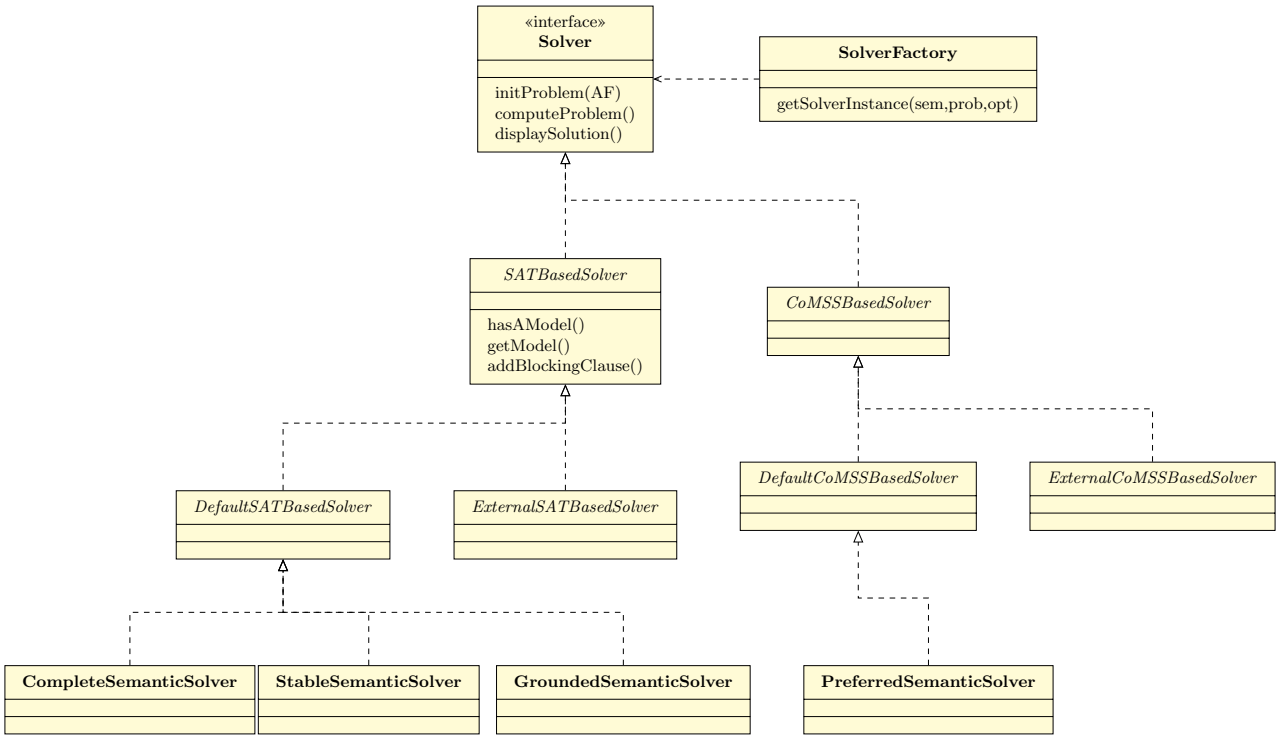

FIGURE 3 – Diagramme de classe simplifié de la partie « solveur » de CoQuiAAS

décompose en une famille de 20 instances dites `real`, dont le nombre d'arguments va de 5000 à 100000, et 79 instances aléatoires dont le nombre d'arguments varie entre 20 et 1000. Le deuxième jeu d'instances comporte des instances aléatoires dont le nombre d'arguments varie entre 200 et 400. CoQuiAAS a été exécuté sur des machines équipées de processeurs Intel Xeon cadencés à 3.00 GHz associés à 2 Go de mémoire vive, dont le système d'exploitation est la distribution GNU/Linux CentOS 6.0 (64 bits).

Nous avons en priorité étudié l'efficacité pratique de notre méthode sur les problèmes d'énumération, étant donné que le temps pour énumérer les extensions d'un système d'argumentation est une borne supérieure du temps requis pour les autres tâches que nous avons implémentées. Les résultats sont donnés en Table 2. Nous avons agrégé les temps par famille d'instances, et présentons ici les temps moyens, arrondis à 10^{-2} près. Le symbole – indique que l'intégralité de la famille a atteint le *timeout* fixé à 900 secondes. Les autres familles ont été intégralement résolues.

Nos expérimentations montrent l'efficacité de notre approche sur les instances de la compétition. Seules les instances `rdm1000` ont atteint le *timeout* sans être résolues, pour les sémantiques stable, complète et préférée. Les temps moyens pour la résolution des instances `XXX300` avec ces trois sémantiques sont nettement plus élevés que les temps moyens requis pour les instances `XXX400` sur les mêmes sémantiques. Cela s'explique par la présence de quelques instances particulièrements difficiles dans cette famille. Certaines d'entre elles nécessitent plusieurs dizaines de secondes, et même jusque 280 secondes pour la sémantique complète et 653 secondes pour la sémantique préférée ; cependant la grande majorité des instances de cette famille est tout de même résolue très rapidement (41 instances sont résolues en strictement moins d'une seconde avec la sémantique complète, et 37 instances pour la sémantique préférée).

Famille	#Inst.	Gr	St	Pr	Co
`rdm20`	25	< 0.01	0.01	< 0.01	< 0.01
`rdm50`	24	< 0.01	< 0.01	< 0.01	< 0.01
`rdm200`	24	< 0.01	0.5	5.32	1.57
`rdm1000`	6	0.25	–	–	–
`real`	20	7.25	7.51	8.55	6.88
`XXX200`	4	< 0.01	0.08	0.04	0.03
`XXX300`	64	0.02	12.53	34.8	21.36
`XXX400`	22	0.01	0.12	0.13	0.08

TABLE 2 – Temps moyen par famille pour énumérer les extensions pour les différentes sémantiques

6 Travaux connexes

Plusieurs approches similaires ont été développées ces dernières années. ASPARTIX [21], tout d'abord, propose une implémentation basée sur des techniques ASP pour calculer les extensions d'un système d'argumentation, pour un grand nombre de sémantiques. Il ne permet par contre pas de travailler avec des requêtes comme l'acceptation crédule et l'acceptation sceptique d'un argument. CEGARTIX [18], basé sur SAT, se concentre sur les requêtes dont la complexité est au second niveau de la hiérarchie polynomiale. Bien que performante, la version actuelle de ce logiciel est nettement moins générale que notre bibliothèque, et qui permet déjà de travailler avec toutes les requêtes et sémantiques usuelles, et qui peut être aisément étendue pour travailler avec les autres sémantiques. ArgSemSAT [12], enfin, est aussi basé sur SAT et permet l'énumération des extensions pour les sémantiques usuelles. Pour autant que nous sachions, aucun de ces logiciels ne permet l'intégration aisée d'autres types de solveurs de contraintes.

7 Conclusion

Dans cet article, nous présentons notre approche basée sur SAT et sur l'extraction de MSS dans le formalisme Partial Max-SAT pour résoudre les problèmes d'inférence les plus courants en argumentation abstraite. Nous mettons en avant la conception de notre bibliothèque CoQuiAAS, qui a été développée de façon à être aisément maintenable et évolutive. Une première version de notre logiciel est d'ors et déjà disponible en ligne à l'adresse `http://www.cril.univ-artois.fr/coquiaas`.

Plusieurs pistes sont envisagées pour de futurs travaux. Tout d'abord, concernant l'inférence à partir d'un système d'argumentation abstrait, il existe un certain nombre de sémantiques qui ont été proposées après les quatre sémantiques « classiques » de Dung. Développer une approche pour ces sémantiques est un défi particulièrement intéressant, notamment la sémantique semi-stable [11] et la sémantique stage [27], pour lesquelles même l'acceptation crédule est au deuxième niveau de la hiérarchie polynomiale. Toujours dans le cadre de Dung, nous savons que lorsque certaines propriétés sont satisfaites par le graphe, certaines sémantiques coïncident. Par exemple, si le système d'argumentation est dit « cohérent » (suivant la définition donnée dans [15]), alors la sémantique stable et la sémantique préférée coïncident. Ainsi, il n'est pas nécessaire dans ce cas d'utiliser l'approche à base de MSS pour calculer les extensions préférées, il est possible d'utiliser l'approche pour la sémantique stable, qui est basée sur SAT et est beaucoup plus efficace. De même, il existe une propriété permettant de déterminer si toutes les sémantiques coïncident, ce qui permet de se limiter à la sémantique de base pour raisonner. Nous souhaitons donc développer des méthodes de *pre-processing* afin de permettre de détecter les graphes pour lesquels il est possible d'utiliser une approche dont la complexité est moins élevée.
Nous envisageons aussi d'étendre notre travail aux diverses extensions du cadre de Dung, tels les Weighted Argumentation Frameworks [16], les Preference-based Argumentation Frameworks [1] ou les Value-based Argumentation Frameworks [6].
Enfin, du point de vue de l'expérience de l'utilisateur, il serait très intéressant d'avoir un moyen de visualiser les systèmes d'argumentation et les résultats des requêtes effectuées sur ces systèmes. Nous envisageons donc de développer une interface graphique, basée sur le moteur de calculs de CoQuiAAS, pour améliorer la qualité des interactions entre l'utilisateur et le système.

Remerciements

Nous tenons à remercier les relecteurs pour leur travail et les commentaires qu'ils ont exprimés.
Ce travail a bénéficié d'une aide de l'Agence Nationale de la Recherche portant la référence ANR-13-BS02-0004 dans le cadre du projet AMANDE.
Ce travail est financé en partie par le conseil régional Nord-Pas de Calais et le programme FEDER.

Références

[1] Leila Amgoud and Claudette Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3) :197–215, 2002.

[2] Leila Amgoud and Caroline Devred. Argumentation frameworks as constraint satisfaction problems. *Annals of Mathematics and Artificial Intelligence*, 69(1) :131–148, 2013.

[3] Leila Amgoud and Nabil Hameurlain. An argumentation-based approach for dialog move selection. In *3rd International Workshop on Argumentation in Multi-Agent Systems, ArgMAS 2006*, pages 128–141, 2006.

[4] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental SAT solving with assumptions : Application to MUS extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing - SAT 2013*, pages 309–317, 2013.

[5] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using

hitting set dualization. In *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages, PADL 2005*, pages 174–186, 2005.

[6] Trevor J. M. Bench-Capon. Value-based argumentation frameworks. In *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning, NMR 2002*, pages 443–454, 2002.

[7] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning, NMR 2004*, pages 59–64, 2004.

[8] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.

[9] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[10] Martin Caminada and Mikolaj Podlaszewski. Grounded semantics as persuasion dialogue. In *Proceedings of the 4th International Conference on Computational Models of Argument, COMMA 2012*, pages 478–485, 2012.

[11] Martin W. A. Caminada, Walter Alexandre Carnielli, and Paul E. Dunne. Semi-stable semantics. *Journal of Logic and Computation*, 22(5) :1207–1254, 2012.

[12] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. Argsemsat : Solving argumentation problems using SAT. In *Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014*, pages 455–456, 2014.

[13] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[14] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2005*, pages 317–328, 2005.

[15] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence*, 77(2) :321–357, 1995.

[16] Paul E. Dunne, Anthony Hunter, Peter McBurney, Simon Parsons, and Michael Wooldridge. Weighted argument systems : Basic definitions, algorithms, and complexity results. *Artificial Intelligence*, 175(2) :457–486, 2011.

[17] Paul E. Dunne and Michael Wooldridge. Complexity of abstract argumentation. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, chapter 5, pages 85–104. Springer, 2009.

[18] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. CEGARTIX : A SAT-Based Argumentation System. Pragmatics of SAT Workshop, PoS 2012, 2012.

[19] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing - SAT 2003*, pages 502–518, 2003.

[20] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, 89(4) :543–560, 2003.

[21] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Aspartix : Implementing argumentation frameworks using answer-set programming. In *Proceedings of the 24th International Conference on Logic Programming, ICLP 2008*, 2008.

[22] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. An experimentally efficient method for (MSS, CoMSS) partitioning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2666–2673, 2014.

[23] Jean-Marie Lagniez and Armin Biere. Factoring out assumptions to speed up MUS extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing - SAT 2013*, pages 276–292, 2013.

[24] João Leite and João Martins. Social abstract argumentation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, pages 2287–2292, 2011.

[25] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1) :1–33, 2008.

[26] João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, 2013.

[27] Bart Verheij. Two approaches to dialectical argumentation : Admissible sets and argumentation stages. In *Proceedings of the biannual International Conference on Formal and Applied Practical Reasoning, FAPR 1996*, pages 357–368. Universiteit, 1996.