Argumentation et CP-jeux booléens

E. Bonzon[⋆] elise.bonzon@mi.parisdescartes.fr
C. Devred[†] caroline.devred@info.univ-angers.fr
M-C. Lagasquie-Schiex[‡] lagasq@irit.fr

[⋆]LIPADE, Université Paris Descartes
45 rue des Saints Pères
75006 Paris, France

[†]LERIA, Université d'Angers
2 Boulevard Lavoisier
49045 Angers Cedex 01, France

[‡]IRIT, Université Paul Sabatier
118 route de Narbonne
F-31062 Toulouse Cedex 9, France

Résumé :
Des liens entre argumentation et théorie des jeux existent déjà. Par exemple, les jeux dynamiques permettent de simuler les interactions entre agents dans un processus d'argumentation. Dans ce papier, nous établissons un nouveau lien entre ces domaines dans un cadre statique : nous montrons comment un CP-jeu booléen peut être utilisé pour calculer des extensions en argumentation, et donnons des algorithmes formels pour le faire.

Mots-clés : Argumentation, théorie des jeux

Abstract:
There already exist some links between argumentation and game theory. For instance, dynamic games can be used for simulating interactions between agents in an argumentation process. In this paper, we establish a new link between these domains in a static framework : we show how CP-Boolean games can be used for computing extensions of argumentation semantics, and give formal algorithms to do so.

Keywords: Argumentation, game theory

1 Introduction

Depuis quelques années, l'argumentation est devenue une approche incontournable pour résoudre des problèmes en intelligence artificielle ; elle peut servir, entre autres, pour faire du raisonnement par défaut ou pour simuler certaines formes de dialogue entre agents (voir par exemple [21, 22, 4, 1, 10]).
Dans ce cadre, l'argumentation correspond à l'étude de l'échange d'arguments entre agents en interaction, cette interaction prenant habituellement la forme d'un conflit (appelé "attaque"). Si on se place par exemple dans un cadre logique, un argument peut alors être vu comme une paire ⟨ensemble d'hypothèses, conclusion⟩, où l'ensemble des hypothèses supporte la conclusion selon certains schémas d'inférence logique, et il y aura conflit si, par exemple, la conclusion d'un argument contredit l'hypothèse d'un autre argument.

La principale difficulté pour toute théorie argumentative est la sélection d'un ensemble acceptable d'arguments, basée sur les interactions entre arguments. Intuitivement, un ensemble acceptable d'arguments doit être dans un certain sens "cohérent" (par exemple ne contenant pas d'arguments en conflit) et "assez fort" (capable par exemple de se défendre contre tous les arguments qui l'attaquent). Il est plus facile d'étudier cette notion d'acceptabilité en utilisant des systèmes d'argumentation abstraits (SA), comme ceux proposés par Dung ([16]) ; dans ces systèmes, la notion d'abstraction repose sur le fait que les arguments et leurs interactions sont donnés respectivement sous la forme d'un ensemble d'arguments et d'une relation binaire sur cet ensemble représentant l'attaque (on ne se préoccupe donc ni de la nature des arguments, ni de la manière dont les interactions sont construites). Notons toutefois que, même dans le cadre abstrait de Dung, la complexité des problèmes associés reste prohibitif dans le cas général (par exemple, "vérifier si un ensemble d'arguments est une extension préférée" est un problème coNP-complet).

D'autre part, la théorie des jeux tente d'analyser formellement les interactions stratégiques entre agents. Intuitivement, un jeu non coopératif est composé d'un ensemble d'agents (ou joueurs), et, pour chaque agent, d'un ensemble de stratégies possibles et d'une fonction d'utilité qui associe toutes les combinaisons possibles de stratégies à une valeur numérique (dans cet article, nous ne considérons que des jeux *statiques*, dans lesquels les agents choisissent leurs stratégies en parallèle, sans observer les choix des autres joueurs).
Un des principaux problèmes de cette approche est la difficulté d'exprimer efficacement[1] la

[1]Car le nombre de combinaisons possibles de stratégies est expo-

fonction d'utilité. Une solution à ce problème consiste à représenter les préférences des agents en utilisant un langage de représentation *structuré* et *compact*. Les jeux booléens permettent de répondre à ce problème : chaque agent contrôle un ensemble de variables *booléennes* (binaires), tandis qu'une formule propositionnelle permet de représenter ses préférences (voir [20, 19]). Ce cadre reste un peu limité (à cause de la dichotomie de préférences, une formule ne pouvant être que vraie ou fausse), mais il peut facilement être étendu en remplaçant les formules propositionnelles par des *CP-nets* (voir [5]). Il existe dans ce contexte quelques résultats de complexité intéressants (par exemple, "vérifier si un profil de stratégies est un équilibre de Nash en stratégies pures" est un problème polynomial lorsque tous les CP-nets sont acycliques).

Des liens entre argumentation et théorie des jeux ont déjà été identifiés ; par exemple, dans [16, 9], des jeux sont utilisés pour définir la théorie de la preuve et les algorithmes associés à certains problèmes d'acceptabilité en argumentation. Toutefois, les jeux considérés sont toujours dynamiques, et il n'y a pas de travaux spécifiques concernant les jeux statiques et l'argumentation. L'objectif premier de cet article est donc de proposer une traduction d'un système d'argumentation vers un jeu statique (un CP-jeu booléen) ; cela nous permettra, dans un second temps, de calculer les extensions préférées en utilisant les concepts bien connus de la théorie des jeux (équilibres de Nash en stratégies pures : PNE). Ainsi nous établirons un nouveau lien entre argumentation et jeux, tout en fournissant une nouvelle méthode de calcul des extensions préférées (même si cette nouvelle méthode n'est pas plus efficace que les algorithmes existants – voir par exemple [14, 17, 18, 9]).

Cet article est organisé comme suit : le cadre d'argumentation de Dung et les CP-jeux booléens sont présentés brièvement en section 2 et section 3. La section 4 présente le noyau de ce travail : comment traduire un système d'argumentation en un CP-jeu booléen, et comment utiliser ce jeu pour calculer les extensions de la sémantique argumentative. Quelques travaux connexes sont ensuite présentés en section 5, puis la conclusion est donnée en section 6.

nentiel en nombre de joueurs et en nombre de variables contrôlées par chaque joueur.

2 Systèmes d'argumentation (SA)

Dung a proposé dans [16] un cadre abstrait pour l'argumentation, dans lequel il s'intéresse principalement aux définitions des statuts des arguments. Pour cela, il suppose qu'un ensemble d'arguments est donné, ainsi que différents conflits entre ces derniers. Nous rappelons brièvement ce cadre :

Définition 1 *Un* système d'argumentation (SA) *est une paire* $\langle \mathcal{A}, \mathcal{R} \rangle$ *composée d'un ensemble d'arguments* $\mathcal{A}$ *et d'une relation binaire* $\mathcal{R}$ *sur* $\mathcal{A}$ *appelée* relation d'attaque. $\forall a_i, a_j \in \mathcal{A}$, $a_i \mathcal{R} a_j$ *signifie que* a_i attaque a_j *(ou que* a_j est attaqué par a_i*). Un SA peut être représenté par un graphe orienté appelé le* graphe d'interaction, *dont les nœuds sont des arguments, et les arcs représentent les relations d'attaque.*

Dans le cadre de Dung, l'*acceptabilité d'un argument* dépend de son appartenance à des ensembles appelés extensions qui doivent vérifier certaines propriétés caractéristiques représentant l'acceptabilité collective. Soit $\mathsf{SA} = \langle \mathcal{A}, \mathcal{R} \rangle$, un système d'argumentation, et $S \subseteq \mathcal{A}$ un ensemble d'arguments. Les principales propriétés caractéristiques sont :

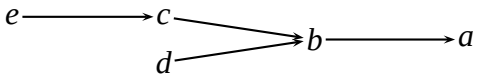
Définition 2 *S est* sans conflit *pour* SA *si et seulement s'il n'existe pas de* a_i, a_j *dans S tels que* $a_i \mathcal{R} a_j$.
Un argument a est acceptable pour *S dans* SA *ssi* $\forall b \in \mathcal{A}$ *tel que* $b \mathcal{R} a$, $\exists c \in S$ *tel que* $c \mathcal{R} b$. *S est* acceptable *pour* SA *ssi* $\forall a \in S$, *a est acceptable pour S dans* SA.

Plusieurs *sémantiques pour l'acceptabilité* ont alors été définies dans [16]. Par exemple :

Définition 3 *S est un* ensemble admissible *de* SA *ssi S est sans conflit et acceptable pour* SA.
S est une extension préférée *de* SA *ssi S est* $\subseteq$*-maximal parmi les ensembles admissibles de* SA.
S est une extension stable *de* SA *ssi S est sans conflit et S attaque tout argument qui n'appartient pas à S.*

Ces notions sont illustrées sur les exemples suivants.

Exemple 1 *Soit* $\mathsf{SA} = \langle \{a,b,c,d,e\}, \{(b,a), (c,b), (d,b), (e,c)\} \rangle$ *représenté par :*



$\{e,c\}$ n'est pas sans conflit ; b n'est pas acceptable pour $\{e\}$; a est acceptable pour $\{e,d\}$; $\{d,a\}$ est un ensemble admissible et $\{e,d,a\}$ est l'extension stable et préférée de SA.

Nous aurons besoin des propriétés suivantes ([16, 17, 18, 15]) :

Proposition 1 *Soit* $\mathsf{SA} = \langle \mathcal{A}, \mathcal{R} \rangle$ *tel que* $\mathcal{A} \neq \varnothing$.

1. *Chaque argument non attaqué appartient à toute extension préférée de* SA *(voir [16]).*
2. *Un système d'argumentation* SA *acyclique contient seulement une extension préférée (voir [17, 18]).*
3. *Si $\varnothing$ est l'unique extension préférée, alors* SA *contient au moins un circuit de longueur impaire (voir [17, 18, 15]).*
4. *Si* SA *ne contient aucun circuit de longueur impaire, alors ses extensions préférées ne sont pas vides (voir [17, 18, 15]).*
5. *Si* SA *ne contient aucun circuit de longueur impaire, alors chaque extension préférée est également stable*[2] *(voir [17, 18, 15]).*

De nombreux travaux dans le domaine de l'argumentation utilisent l'hypothèse simplificatrice suivante : les systèmes d'argumentation ne contiennent pas de circuits de longueur impaire, de tels circuits étant considérés comme des paradoxes (car ils généralisent des situations dans lesquelles un argument s'attaque lui-même). Nous adopterons ici ce point de vue[3] en considérant que *les systèmes d'argumentation traduits ne doivent pas contenir de circuits de longueur impaire*[4] ; par contre, les systèmes d'argumentation contenant des circuits de longueur paire seront pris en compte.
Notons que, dans le cas où il n'existe pas de circuits de longueur impaire, le système d'argumentation présente des propriétés particulièrement intéressantes :

Conséquence 1 *Soit* $\mathsf{SA} = \langle \mathcal{A}, \mathcal{R} \rangle$ *un SA sans circuit de longueur impaire. Soit E une extension préférée de* SA*, et soit a un argument de* SA*. Si E ne contient pas d'attaquant de a, alors $a \in E$.*

[2] Le système d'argumentation est alors dit "cohérent" (voir [16]).

[3] Même si certains circuits de longueur impaire peuvent avoir du sens.

[4] Et, si ce n'est pas le cas, ces circuits de longueur impaire seront supprimés du système d'argumentation avant traduction.

Preuve : Nous pouvons appliquer la proposition 1.5 issue de [17, 18, 15]. E est donc une extension stable. Donc, comme E est stable, si on suppose que $a \notin E$ alors $\exists b \in E$ tel que $b \mathcal{R} a$. Mais ceci est impossible car il n'existe pas d'attaquant de a dans E. Donc $a \in E$. ■

3 CP-jeux booléens

Nous allons tout d'abord introduire quelques notations qui nous seront utiles. Soit V un ensemble fini de variables propositionnelles et L_V le langage propositionnel construit à partir de V, des connecteurs usuels et des constantes booléennes $\top$ (*vrai*) et $\bot$ (*faux*). Les formules de L_V sont dénotées par φ, ψ, etc. 2^V est l'ensemble des interprétations pour V, avec la convention usuelle établissant que pour $M \in 2^V$ et $x \in V$, M donne la valeur *vrai* à x si $x \in M$ et *faux* sinon. Soit $X \subseteq V$. 2^X est l'ensemble des *X-interprétations*. Les X-interprétations sont dénotées par la liste de toutes les variables de X, associées au symbole $\overline{\ }$ lorsque la variable est mise à faux. Par exemple, si $X = \{a,b,d\}$, la X-interprétation $M = \{a,d\}$ est notée $a\overline{b}d$. Une *relation de préférences* $\succeq$ est une relation binaire (non nécessairement complète) réflexive et transitive sur 2^V. Soit $M, M' \in 2^V$. La relation de préférences stricte $\succ$ associée à $\succeq$ est définie par $M \succ M'$ ssi $M \succeq M'$ et non $M' \succeq M$.

3.1 CP-nets

Nous étudions dans cette section un langage compact de représentation de préférences sur des domaines combinatoires très populaire : les CP-nets. Ce modèle graphique exploite l'indépendance préférentielle conditionnelle afin de structurer les préférences d'un agent sous l'hypothèse *ceteris paribus* [6, 7]. Si les CP-nets sont généralement construits à partir de variables ayant un domaine arbitraire fini, nous allons ici utiliser des CP-nets "propositionnels", c'est-à-dire des CP-nets avec des variables binaires.

Définition 4 *Soit V un ensemble de variables propositionnelles et $\{X,Y,Z\}$ une partition de V. X est* conditionnellement préférentiellement indépendant *de Y étant donné Z ssi $\forall z \in 2^Z$, $\forall x_1, x_2 \in 2^X$ et $\forall y_1, y_2 \in 2^Y$ nous avons : $x_1 y_1 z \succeq x_2 y_1 z$ ssi $x_1 y_2 z \succeq x_2 y_2 z$.*

Pour chaque variable X, l'agent spécifie un ensemble de *variables parents* $Pa(X)$ qui peuvent affecter ses préférences sur les valeurs de X. Formellement, X et $V \setminus (\{X\} \cup Pa(X))$ sont conditionnellement préférentiellement indépendants étant donné $Pa(X)$. Cette information est utilisée pour créer le CP-net :

Définition 5 *Soit V un ensemble de variables propositionnelles. $\mathcal{N} = \langle \mathcal{G}, \mathcal{T} \rangle$ est un* CP-net *sur V, où $\mathcal{G}$ est un graphe orienté sur V, et $\mathcal{T}$ est un ensemble de tables de préférences conditionnelles $CPT(X_j)$ pour chaque $X_j \in V$. Chaque $CPT(X_j)$ associe un ordre linéaire $\succ_p^j$ à chaque instanciation $p \in 2^{Pa(X_j)}$.*

Les informations représentées par un CP-net $\mathcal{N}$ peuvent être vues comme un ensemble d'assertions logiques sur les préférences d'un agent concernant un ensemble complet d'assignations de variables dans un graphe. Ces préférences ne sont pas totales : elles ne permettent pas en général de déterminer une unique relation de préférences.

Exemple 2 *Soit le CP-net donné en Figure 1 à propos de préférences sur un dîner. Les variables S et V correspondent respectivement à la soupe et le vin. Je préfère strictement manger de la soupe de poisson (S_p) plutôt que de la soupe de légumes (S_l), tandis que mes préférences sur le vin dépendent de la soupe que je mange : je préfère du vin rouge (V_r) avec la soupe de légumes ($S_l : V_r \succ V_b$), et du vin blanc (V_b) avec la soupe de poisson ($S_p : V_b \succ V_r$). Donc, $D(S) = \{S_p, S_l\}$ et $D(V) = \{V_r, V_b\}$.*

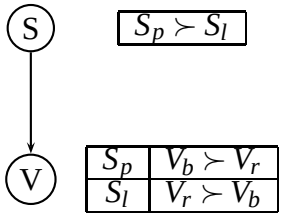


FIG. 1 – CP-net "Dîner"

La figure 2 représente la relation de préférences induite par ce CP-net. L'élément du bas ($S_l \wedge V_b$) est le pire cas, tandis que l'élément du haut ($S_p \wedge V_b$) est le meilleur cas.

Il y a un arc entre les nœuds $(S_p \wedge V_b)$ et $(S_l \wedge V_b)$ car il est possible de comparer ces états toutes choses étant égales par ailleurs.

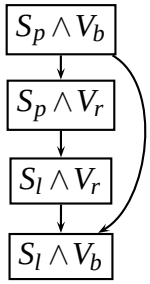


FIG. 2 – Graphe de préférences induit par le CP-net "Dîner"

Nous pouvons donc complètement ordonner les états possibles (du préféré au moins bon) :

$$(S_p \wedge V_b) \succ (S_p \wedge V_r) \succ (S_l \wedge V_r) \succ (S_l \wedge V_b)$$

Cette relation $\succ$ est la seule qui satisfasse ce CP-net.

3.2 CP-jeux booléens

Les jeux booléens [20, 19] permettent de représenter de façon compacte des jeux statiques à 2 joueurs ayant des préférences binaires. Dans [5], les jeux booléens sont généralisés avec des préférences non dichotomiques : ils sont couplés avec des CP-nets propositionnels.

Définition 6 *Un* CP-jeu booléen *est un 4-tuple $G = (N, V, \pi, \Phi)$, où N est un ensemble de joueurs, V est un ensemble de variables propositionnelles, $\pi : N \mapsto V$ est une fonction d'affectation de contrôle qui définit une partition de V, et $\Phi = \langle \mathcal{N}_1, \ldots, \mathcal{N}_n \rangle$, chaque $\mathcal{N}_i$ étant un CP-net sur V dont le graphe est dénoté par $\mathcal{G}_i$, et $\forall i \in N$, $\succeq_i = \succeq_{\mathcal{N}_i}$.*

La fonction d'affectation de contrôle π associe à chaque joueur les variables qu'il contrôle, chaque variable étant contrôlée par un et un seul agent,[5] *i.e.*, $\{\pi_1, \ldots, \pi_n\}$ forme une partition de V.

Définition 7 *Soit $G = (N, V\pi, \Phi)$ un CP-jeu booléen. Une* stratégie *s_i pour le joueur i est une π_i-interprétation. Un* profil de stratégies *s est un n-tuple $s = (s_1, \ldots, s_n)$ où pour chaque i, $s_i \in 2^{\pi_i}$.*

En d'autres mots, une stratégie pour *i* est une assignation de valeurs de vérité à chaque variable

[5] L'ensemble de toutes les variables contrôlées par *i* sera noté π_i plutôt que $\pi(i)$.

contrôlée par i. Comme $\{\pi_1,\ldots,\pi_n\}$ forme une partition de V, un profil de stratégies définit une interprétation (non ambiguë) pour V. Par abus de notation, nous noterons $s \in 2^V$ pour représenter la valeur assignée par s à des variables.

Nous utiliserons dans la suite de ce papier les notations suivantes, qui sont classiques en théorie des jeux : soit $G=(N,V,\pi,\Phi)$ un CP-jeu booléen avec $N=\{1,\ldots,n\}$, et $s=(s_1,\ldots,s_n)$, $s'=(s'_1,\ldots,s'_n)$ deux profils de stratégies. s_{-i} dénote la projection de s sur $N\setminus\{i\}$: $s_{-i}=(s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_n)$; similairement, π_{-i} dénote l'ensemble des variables contrôlées par tous les joueurs sauf i : $\pi_{-i}=V\setminus\pi_i$; enfin, (s'_i,s_{-i}) représente le profil de stratégies obtenu à partir de s en remplaçant s_i par s'_i sans modifier les autres stratégies : $(s'_i,s_{-i})=(s_1,\ldots,s_{i-1},s'_i,s_{i+1},\ldots,s_n)$.

Un équilibre de Nash en stratégies pures (PNE) est un profil de stratégies tel que la stratégie de chaque joueur est une réponse optimale aux stratégies des autres joueurs. Les PNEs sont classiquement définis pour des jeux dans lesquels les préférences sont complètes, ce qui n'est pas nécessairement le cas ici. Nous devons donc introduire la notion de PNE *fort*.

Définition 8 *Soit $G=(N,V,\pi,\Phi)$ et $Pref_G=\langle\succeq_1,\ldots,\succeq_n\rangle$ la collection des relations de préférences sur 2^V induite par Φ. Soit $s=(s_1,\ldots,s_n)\in 2^V$. s est un* PNE fort *(SPNE) pour G ssi $\forall i\in\{1,\ldots,n\}$, $\forall s'_i\in 2^{\pi_i}$, $(s'_i,s_{-i})\preceq_i(s_i,s_{-i})$.*

La proposition suivante a été prouvée dans [5] :

Proposition 2 *Soit $G=(N,V,\pi,\Phi)$ un CP-jeu booléen tel que les graphes $\mathcal{G}_i$ sont tous identiques ($\forall i,j\in N$, $\mathcal{G}_i=\mathcal{G}_j$) et acycliques. G a alors un et un seul PNE fort.*

La preuve de ce résultat utilise la *forward sweep* procédure [6] (cette procédure consiste à instancier les variables selon un ordre compatible avec le graphe, en choisissant pour chaque variable sa valeur préférée en fonction de l'instanciation de ses parents). De plus, il a été montré dans [5] que ce SPNE peut être construit en temps polynomial.

4 Argumentation et CP-jeux booléens

Notre objectif ici est de transformer un SA en un CP-jeu booléen G, puis d'utiliser les outils bien connus de la théorie des jeux, et plus particulièrement les propriétés spécifiques des CP-jeux booléens, afin de trouver les extensions préférées du SA. Ce travail nous permet de créer de nouveaux liens entre jeux et argumentation.

4.1 Traduction d'un système d'argumentation en un CP-jeu booléen

Cette transformation est faite par l'algorithme 1. Ce dernier suppose l'existence de deux autres algorithmes :

– ESTCYCLIQUE qui retourne *vrai* s'il existe au moins un circuit dans le graphe d'argumentation[6],
– SUPCIRCUITSIMP pour effacer tous les circuits de longueur impaire s'il en existe dans le système d'argumentation[7].

L'exécution de ces deux algorithmes peut être vue comme une étape de pré-compilation de l'algorithme 1. Comme l'algorithme ESTCYCLIQUE ne détecte pas directement les circuits de longueur impaire, il peut sembler inutile dans la pré-compilation de l'algorithme 1. Pourtant, comme ESTCYCLIQUE est un algorithme linéaire tandis que SUPCIRCUITSIMP est polynomial, nous pensons qu'il est intéressant d'éviter une exécution inutile de SUPCIRCUITSIMP lorsque SA est acyclique.

[6] Cet algorithme est linéaire :

(Étape 1) effacer tous les nœuds qui n'ont pas de prédécesseurs ;

(Étape 2) recommencer l'étape 1 jusqu'à ce que, soit tous les nœuds restants aient au moins un prédécesseur (le graphe initial contient un circuit), soit le graphe est vide (donc le graphe initial ne contient pas de circuit).

[7] Cet algorithme est polynomial :

(Étape 1) calcul de la matrice d'adjacence booléenne correspondant à tous les chemins d'attaque de longueur impaire minimaux ; il est suffisant de prendre la matrice d'adjacence booléenne $\mathcal{M}$ ($\mathcal{M}(i,j)=1$ s'il existe un arc de i à j dans SA) et de calculer $\mathcal{M}^{\text{olc}}=\mathcal{M}^1+\mathcal{M}^3+\ldots+\mathcal{M}^{2n-1}$ avec $n=|\mathcal{A}|$ (la limite $2n-1$ est obtenue en utilisant un résultat général de la théorie des graphes : si un graphe orienté contient un chemin de a vers b, il existe alors un chemin simple – un chemin dans lequel chaque nœud n'apparaît qu'une seule fois – de a vers b) ;

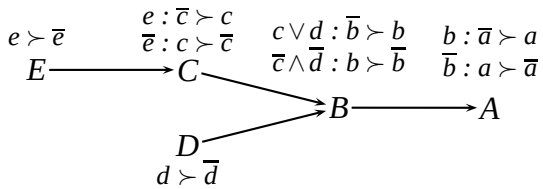
(Étape 2) effacer tous les arguments pour lesquels l'élément diagonal de $\mathcal{M}^{\text{olc}}$ vaut 1 ;

(Étape 3) effacer tous les arcs ayant un argument éliminé en point de départ ou d'arrivée.

Soit SA un système d'argumentation qui ne contient pas de circuit de longueur impaire. Les principes de l'algorithme 1 sont les suivants :
- chaque argument de SA est une variable de G ;
- chaque variable est contrôlée par un joueur différent (nous avons donc autant de joueurs que de variables) ;
- les CP-nets de chaque joueur sont définis de façon identique :
  - le graphe du CP-net est exactement le graphe orienté du SA ;
  - les préférences sur chaque variable v qui n'est pas attaquée sont $v \succ \overline{v}$ (si un argument n'est pas attaqué, nous voulons le protéger ; la valeur *vrai* de la variable v est donc préférée à sa valeur *faux*),
  - les préférences sur chaque variable v qui est attaquée par un ensemble de variables $\mathcal{R}^{-1}(v)$ dépend de ces variables : si au moins une variable $w \in \mathcal{R}^{-1}(v)$ est satisfaite, v ne peut pas être satisfaite (nous avons donc $\bigvee_{w\in\mathcal{R}^{-1}(v)} w : \overline{v} \succ v$[8]) ; par contre, si aucune variable $w \in \mathcal{R}^{-1}(v)$ n'est satisfaite, v peut être satisfaite (et donc $\bigwedge_{w\in\mathcal{R}^{-1}(v)} \overline{w} : v \succ \overline{v}$).

Le CP-jeu booléen G est construit en temps polynomial à partir d'un système d'argumentation SA (même si SA est cyclique et que nous ayons d'abord à effacer les circuits de longueur impaire).

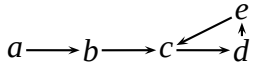
Exemple 3 *Soit* $\mathsf{SA} = \langle\{a,b,c,d,e\},\ \{(b,a),\ (c,b),\ (d,b),(e,c)\}\rangle$ *(*SA *est acyclique). Sa traduction en un CP-jeu booléen* $G = (N,V,\pi,\Phi)$ *s'obtient en appliquant l'algorithme 1 :* $V = \{a,b,c,d,e\}$ *et* $N = \{1,2,3,4,5\}$, *avec* $\pi_1 = \{a\}$, $\pi_2 = \{b\}$, $\pi_3 = \{c\}$, $\pi_4 = \{d\}$ *et* $\pi_5 = \{e\}$. *Le CP-net suivant représente les préférences de tous les joueurs*[9] *:*



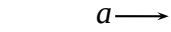
Exemple 4 *Soit* $\mathsf{SA} = \langle\{a,b\},\{(a,b),\ (b,a)\}\rangle$. *On obtient en appliquant l'algorithme 1 :* $V = \{a,b\}$ *et* $N = \{1,2\}$, *avec* $\pi_1 = \{a\}$, $\pi_2 = \{b\}$ *(*SA *est cyclique mais ne contient que des circuits pairs). Le CP-net suivant représente les préférences de tous les joueurs.*

Exemple 5 *Soit* $\mathsf{SA} = \langle\{a,b,c,d,e\},\ \{(a,b),\ (b,c),\ (c,d),\ (d,e),\ (e,c)\}\rangle$. *Le* SA *initial contient un circuit de longueur impaire, qui doit donc être supprimé. Le* SA *final ne contient que a et b.*

Graphe initial pour le SA



Graphe final pour le SA



En appliquant l'algorithme 1, on obtient donc $V = \{a,b\}$, $N = \{1,2\}$, *avec* $\pi_1 = \{a\}$, $\pi_2 = \{b\}$ *et le CP-net suivant qui représente les préférences de tous les joueurs :*

Algorithme 1 : Traduction d'un système d'argumentation en CP-jeu booléen

début

/* ENTREE : $\mathsf{SA} = \langle\mathcal{A},\mathcal{R}\rangle$ système d'argumentation */
/* SORTIES : $G = (N,V,\pi,\Phi)$ CP-jeu booléen, SA après suppression de tous les circuits impairs */
/* VARIABLES LOCALES : i = agent courant, a = argument courant */

/* suppression de tous les circuits impairs */
si ESTCYCLIQUE(SA) **alors**
  SA = SUPCIRCUITSIMP(SA)
/* SA ne contient plus de circuits impairs*/
/* calcul des CPTs de chaque argument */
pour $a \in \mathcal{A}$ **faire**
  si $\mathcal{R}^{-1}(a) = \varnothing$ **alors** $CPT(a) = a \succ \overline{a}$ /* argument non attaqué */
  sinon /* cas de tous les autres arguments */
    $CPT(a) = \{\bigvee_{v\in\mathcal{R}^{-1}(a)} v : \overline{a} \succ a\} \cup \{\bigwedge_{v\in\mathcal{R}^{-1}(a)} \overline{v} : a \succ \overline{a}\}$
/* calcul du CP-net $\mathcal{N}$ */
$\mathcal{N} = \langle \mathsf{SA}, \cup_{a\in\mathcal{A}} CPT(a)\rangle$ /* graphe d'attaque */
/* après suppression de tous les circuits impairs, */
/* associé au CPT de chaque argument */
/* calcul de N, V, π et Φ */
$i = 1$
$N = \varnothing$
$V = \mathcal{A}$ /* chaque argument est une variable */
pour $a \in \mathcal{A}$ **faire**
  $N = N \cup \{i\}$ /* un agent pour chaque argument */
  $\pi_i = \{a\}$ /* i contrôle seulement cet argument */
  $\mathcal{N}_i = \mathcal{N}$ /* le même CP-net pour chaque agent */
  $i = i + 1$
retourner $(G = (N,V,\pi,\langle\mathcal{N}_1,\ldots,\mathcal{N}_{|V|}\rangle)$, SA)

fin

[8] La formule $w : \overline{v} \succ v$ (resp. $\overline{w} : \overline{v} \succ v$) signifie que pour la valeur *vrai* (resp. *faux*) de la variable w, la valeur *faux* de la variable v est préférée à la valeur *vrai*.

[9] Afin de différencier le CP-net du SA, nous représentons les nœuds du CP-net en majuscule, et ceux du SA en minuscule.

$$a \succ \overline{a}\ A \longrightarrow B \quad \begin{array}{l} a : \overline{b} \succ b \\ \overline{a} : b \succ \overline{b} \end{array}$$

Proposition 3 *Soit* $\mathsf{SA} = \langle \mathcal{A}, \mathcal{R} \rangle$ *un système d'argumentation. Soit* $G = (N, V, \pi, \Phi)$ *le CP-jeu booléen et* SA' *le système d'argumentation obtenus en appliquant l'algorithme 1 sur* SA. *s est une extension préférée de* SA' *ssi s est un SPNE de*[10] *G.*

Preuve : Étudions tout d'abord la direction $\Rightarrow$.

Soit s une extension préférée de SA'. Supposons que s n'est pas un SPNE du CP-jeu booléen associé. Donc, $\exists i \in N$, $\exists s'_i \in 2^{\pi_i}$, $\exists s_{-i} \in 2^{\pi_{-i}}$, tels que $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$. Soit x_i la variable de V telle que $\pi_i = \{x_i\}$ (x_i est également un argument de SA'). Plusieurs cas sont possibles :

- $\mathcal{R}^{-1}(x_i) = \varnothing$ (x_i n'est pas attaquée). D'après l'algorithme 1 nous avons $CPT(x_i) = x_i \succ \overline{x}_i$. Comme $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$, nous savons que $s_i = \overline{x}_i$ ($x_i \notin s$). Mais d'après la proposition 1.1, si $\mathcal{R}^{-1}(x_i) = \varnothing$ alors $x_i \in s$. Nous avons une contradiction.
- $\mathcal{R}^{-1}(x_i) \neq \varnothing$ (x_i a au moins un attaquant). D'après l'algorithme 1, nous avons $CPT(x_i) = \{\bigvee_{w \in \mathcal{R}^{-1}(x_i)} w : \overline{x}_i \succ x_i\} \cup \{\bigwedge_{w \in \mathcal{R}^{-1}(x_i)} \overline{w} : x_i \succ \overline{x}_i\}$. Deux cas sont possibles :
  - $\forall x_j$ tel que $x_j \mathcal{R} x_i$, $x_j \notin s$. Donc, $\bigwedge_{w \in \mathcal{R}^{-1}(x_i)} \overline{w}$ est vrai, et on peut déduire de $CPT(x_i)$ que $x_i \succ \overline{x}_i$. Comme $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$, on a $s_i = \overline{x}_i$ et $s'_i = x_i$. x_i n'est donc pas dans s. Ceci est en contradiction avec la conclusion obtenue en appliquant la conséquence 1 qui dit que $x_i \in s$ (SA' est un système d'argumentation sans circuits impairs, et x_i n'a pas d'attaquant dans l'extension préférée de s). Ce cas est donc impossible.
  - Au moins un argument x_j de $\mathcal{R}^{-1}(x_i)$ appartient à s. $\bigvee_{w \in \mathcal{R}^{-1}(x_i)} w$ est donc vrai, et on peut déduire $\overline{x}_i \succ x_i$ en utilisant $CPT(x_i)$. Comme $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$, on a $s_i = x_i$. Mais ceci est en contradiction avec le fait que s doit être sans conflit ($x_j \in s$ et $x_i \in s$). Ce cas est donc également impossible.

  Aucun cas n'est donc possible si nous supposons que s n'est pas un SPNE. s est donc bien un SPNE.

Étudions à présent la direction $\Leftarrow$.

Soit $s = (s_1, \ldots, s_n)$ un SPNE de G. Nous devons à présent montrer que s est une extension préférée de SA'. Supposons plutôt qu'elle n'en est pas une. Soit $x_i \in s$ une variable telle que (1) ou (2) soient vérifiées avec (1) : $\exists x_j \in s$ tel que $x_i \mathcal{R} x_j$ ou $x_j \mathcal{R} x_i$ (s n'est pas sans conflit) et (2) : $\exists x_j \in \mathcal{A}$ tel que $x_j \mathcal{R} x_i$ et $\nexists x_k \in s$ telle que $x_k \mathcal{R} x_j$ (s n'est pas acceptable).

- s n'est pas sans conflit (1) :
  - $\exists x_j \in s$ tel que $x_i \mathcal{R} x_j$. Comme $x_i \in s$, nous savons que $\bigvee_{w \in \mathcal{R}^{-1}(x_j)} w$, et, en utilisant $CPT(x_j)$, nous pouvons déduire que $\overline{x}_j \succ x_j$. Comme nous savons que s est un SPNE ; pour chaque joueur j qui contrôle x_j, nous avons $\forall s'_j$, $\forall s_{-j}$, $(s_j, s_{-j}) \succ_j (s'_j, s_{-j})$. Donc, $s_j = \overline{x}_j$ et $x_j \notin s$, ce qui est une contradiction.
  - $\exists x_j \in s$ tel que $x_j \mathcal{R} x_i$. Comme $x_i \in s$ et que s est un SPNE, nous savons que $x_i \succ \overline{x}_i$; donc, en utilisant $CPT(x_i)$, nous pouvons en déduire $\bigwedge_{w \in \mathcal{R}^{-1}(x_i)} \overline{w}$. Donc, $x_j \notin s$, ce qui est une contradiction.
- s n'est pas acceptable (2) : $\exists x_j \in \mathcal{A}$ tel que $x_j \mathcal{R} x_i$ et $\nexists x_k \in s$ tel que $x_k \mathcal{R} x_j$. Comme $x_i \in s$ et s est un SPNE, nous savons que $x_i \succ \overline{x}_i$. En utilisant $CPT(x_i)$, nous pouvons en déduire $\bigwedge_{w \in \mathcal{R}^{-1}(x_i)} \overline{w}$. Comme $x_j \mathcal{R} x_i$, nous savons que $x_j \notin s$. Donc, $\overline{x}_j \succ x_j$ et, en utilisant $CPT(x_j)$, nous pouvons en déduire $\bigvee_{w \in \mathcal{R}^{-1}(x_j)} w$. Cela signifie qu'il existe $x_k \in s$ tel que $x_k \in \mathcal{R}^{-1}(x_j)$, ce qui est une contradiction.

Aucun cas n'est donc possible si nous supposons que s n'est pas une extension préférée. s est donc une extension préférée. ∎

Exemple 3 (suite): G a un SPNE $\{ed\overline{c}\overline{b}a\}$ et SA a seulement une extension préférée $\{e, d, a\}$.

Exemple 4 (suite): G a deux SPNEs $\{a\overline{b}, \overline{a}b\}$ et SA a deux extensions préférées $\{a\}$, $\{b\}$.

Exemple 5 (suite): G a un SPNE $\{a\overline{b}\}$ et le SA final (*après suppression des circuits impairs*) a une extension préférée $\{a\}$.

4.2 Calcul des extensions préférées

Comme les extensions préférées correspondent exactement aux SPNEs, les principales propriétés sur le calcul des SPNEs dans les CP-jeux booléens peuvent être appliquées. Le premier

[10] Rappelons que s représente une V-interprétation, et donc si $s = a\overline{b}c$ par exemple, cela correspond à l'ensemble $\{a, c\}$.

cas intéressant concerne les systèmes d'argumentation acycliques :

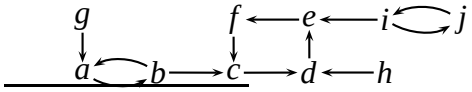
Proposition 4 *Soit* SA *un système d'argumentation. Soit G le CP-jeu booléen et* SA′ *le système d'argumentation obtenus à partir de* SA *en appliquant l'algorithme 1.*
Si SA′ *est acyclique,* SA′ *a alors une et une seule extension préférée qui est calculable en temps polynomial à partir de G.*

Preuve : La transformation de SA en un CP-jeu booléen en appliquant l'algorithme 1 se fait en temps polynomial. Ensuite, le calcul du SPNE de ce jeu se fait en temps polynomial en utilisant la procédure forward sweep (proposition 2 donnée dans [5]), et enfin la proposition 3 montre que ce SPNE correspond exactement à l'extension préférée de SA′ (SA après suppression des circuits impairs). ■

Cette proposition est vraie pour le cas simple des systèmes d'argumentation acycliques. Le calcul de(s) SPNE(s) dans le cas des systèmes d'argumentation cycliques est beaucoup plus compliqué. Les algorithmes 2 et 3 permettent de calculer ces concepts de solution dans le cas où le système d'argumentation contient des circuits de longueur paire.

Ces algorithmes supposent l'existence de l'algorithme CALCCIRCUITINTPOURPROPAG qui renvoie le circuit (ou l'un des circuits s'il y en a plusieurs) qui permet d'atteindre le plus de variables possible dans un ensemble de variables donné[11].

Par exemple, sur le graphe suivant :



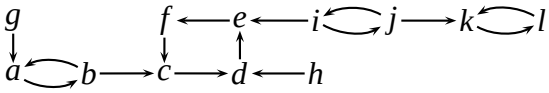
[11] Cet algorithme utilise la matrice booléenne d'adjacence comme l'algorithme SUPCIRCUITSIMP :
- calcul de la matrice booléenne d'adjacence $\mathcal{M}^{\text{ap}}$ qui correspond à tous les chemins minimaux dans le graphe réduit à un ensemble donné de variables $\mathcal{M}^{\text{ap}} = \mathcal{M} + \mathcal{M}^2 + \mathcal{M}^3 + \ldots + \mathcal{M}^{2n}$ avec $n = |V|$; $\mathcal{M}^{\text{ap}}(i)$ représente $(\mathcal{M}^{\text{ap}}(i,1), \ldots, \mathcal{M}^{\text{ap}}(i,n))$;
- `AVoir` $= V$; $C = \varnothing$; `fin`? $=$*faux* ;
- boucle : tant que NON(`fin`?) faire
  $v =$ `top`(`AVoir`) ; `AVoir` $=$ `AVoir` $\setminus \{v\}$;
  si ($\nexists w \in$ `AVoir` t.q.. $\mathcal{M}^{\text{ap}}(v) \subset \mathcal{M}^{\text{ap}}(w)$) alors
  /* pas de variable permettant d'atteindre plus de variables que v */
  $C = C \cup \{v\}$;
  $\forall w \in$ `AVoir` faire si $\mathcal{M}^{\text{ap}}(v) = \mathcal{M}^{\text{ap}}(w)$ alors $C = C \cup \{w\}$;
  `fin`? $=$*vrai* ;
  sinon si `AVoir` est vide alors `fin`? $=$*vrai* ;
- Retourner C

$\{a,b\}$ permet d'atteindre les variables *a*, *b*, *c*, *d*, *e*, *f*, $\{i,j\}$ permet d'atteindre les variables *i*, *j*, *c*, *d*, *e*, *f*. Ces circuits sont plus intéressants que les autres pour propager les valeurs dans le graphe (si ce sont des points de départ pour un processus de propagation, cette propagation sera plus efficace).

Soit $\mathcal{N}$ le CP-net représentant les buts des joueurs d'un CP-jeu booléen. Les principes des algorithmes 2 et 3 sont :

1. instanciation de toutes les variables non attaquées (qui n'ont pas de parents dans $\mathcal{N}$ et sont satisfaites dans le SPNE) ;
2. propagation de ces instanciations autant que possible ;
3. une fois que toutes les instanciations possibles ont été effectuées, répéter les étapes suivantes :
   - si toutes les variables sont instanciées, le SPNE peut être retourné ;
   - sinon :
     - calculer le circuit le plus intéressant *C* avec l'algorithme CALCCIRCUITINTPOURPROPAG (il y en a un sinon toutes les variables auraient été instanciées) ;
     - créer deux nouveaux SPNEs en utilisant l'état courant du SPNE auquel on ajoute pour le premier SPNE une variable de *C* instanciée à *vrai*, pour l'autre SPNE cette même variable instanciée à *faux* ;
     - propager ces instanciations pour chacun de ces SPNEs autant que possible ;
     - reboucler sur l'étape 3.

Exemple 6 *Soit le graphe suivant :*



Les étapes du processus de calcul sont :
- *g et h sont instanciés à vrai (état courant du SPNE* $= gh$*) ;*
- *a et d sont instanciés à faux (état courant SPNE* $= gh\overline{a}\overline{d}$*) ;*
- *b est instancié à vrai (état courant du SPNE* $= gh\overline{a}\overline{d}b$*) ;*
- *c est instancié à faux (état courant du SPNE* $= gh\overline{a}\overline{d}b\overline{c}$*) ;*
- *la simple propagation s'arrête ici. Nous devons maintenant calculer les circuits intéressants dans l'ensemble de variables restant* (e,f,i,j,k,l)*, le résultat est* (i,j) *;*

Algorithme 2 : Calcul des SPNEs d'un CP-jeu booléen obtenu à partir d'un système d'argumentation

début

/* ENTREES : un CP-jeu booléen $G = (N, V, \pi, \Phi)$, où $\Phi = \langle \mathcal{N}_1, \ldots, \mathcal{N}_n \rangle$ */
/* SORTIES : un ensemble de SPNEs *SP* */
/* VARIABLES LOCALES : v = variable courante, In = (resp. Out =) ensemble des variables instanciées à *vrai* (resp. *faux*), R = ensemble des variables restant à instancier */

$In = \varnothing$, $Out = \varnothing$, $R = V$ /* Initialisation */
/* Instanciation de toutes les variables sans parents */
pour $v \in R$ **faire**
  si $Pa(v) = \varnothing$ **alors**
    $R = R \setminus \{v\}$
    $In = In \cup \{v\}$
/* propagation par récursion */
retourner CALCSPNEREC(G, R, In, Out)

fin

Algorithme 3 : CALCSPNEREC : Calcul récursif des SPNEs d'un CP-jeu booléen obtenu à partir d'un système d'argumentation

début

/* ENTREES : un CP-jeu booléen $G = (N, V, \pi, \Phi)$, R = ensemble de variables restant à instancier, In = ensemble de variables instanciées à *vrai*, Out = ensemble de variables instanciées à *faux**/
/* SORTIES : ensemble de SPNEs *SP* */
/* VARIABLES LOCALES : v = variable courante, n = cardinal de R, C = ensemble de variables formant un circuit*/

si $R = \varnothing$ **alors**
  /* variables toutes instanciées, SPNE trouvé */
  retourner $\{(In\overline{Out})\}$
sinon
  $n = |R|$ /* n = nombre de variables restant à instancier */
  pour $v \in R$ **faire**
    /* simple processus de propagation */
    si $Pa(v) \subseteq Out$ **alors**
      /* parents tous instanciés à *faux* */
      $In = In \cup \{v\}$
      $R = R \setminus \{v\}$
    sinon
      si $(Pa(v) \cap In) \neq \varnothing$ **alors**
        /* existe parent(s) instancié à *vrai* */
        $Out = Out \cup \{v\}$
        $R = R \setminus \{v\}$
  si $n = |R|$ **alors**
    /* aucune variable instanciée dans l'instruction Pour */
    C = CALCCIRCUITINTPOURPROPAG(G, R)
    v = TOP(C)
    return(
      CALCSPNEREC(G, $R \setminus \{v\}$, $In \cup \{v\}$, Out)
      $\cup$
      CALCSPNEREC(G, $R \setminus \{v\}$, In, $Out \cup \{v\}$))
  sinon
    /* au moins une variable instanciée dans l'instruction Pour */
    retourner CALCSPNEREC(G, R, In, Out)

fin

– *le processus de propagation repart avec deux SPNEs courants : $gh\overline{a}\overline{d}b\overline{c}i$ et $gh\overline{a}\overline{d}b\overline{c}\overline{i}$;*
– *pour le premier cas, la propagation simple donne $gh\overline{a}\overline{d}b\overline{c}i\overline{e}\,\overline{f}\,jk\overline{l}$, tandis que pour le second cas, nous obtenons $gh\overline{a}\overline{d}b\overline{c}\overline{i}e\,\overline{f}\,jk\overline{l}$. Il y a donc deux SPNEs qui correspondent aux deux extensions préférées $\{g,h,b,e,j,l\}$ et $\{g,h,b,f,i,k\}$.*

La proposition suivante montre que les algorithmes 2 et 3 permettent de calculer exactement l'ensemble des SPNEs du CP-jeu booléen.

Proposition 5 *Soit G un CP-jeu booléen obtenu à partir de l'algorithme 1. Soit SP l'ensemble des profils de stratégies de G donné par les algorithmes 2 et 3. $s \in SP$ ssi s est un SPNE de G.*

Preuve : Étudions tout d'abord la direction $\Rightarrow$.

Soit $s \in SP$. Supposons que s n'est pas un SPNE du CP-jeu booléen associé. Donc, $\exists i \in N$, $\exists s'_i \in 2^{\pi_i}$, $\exists s_{-i} \in 2^{\pi_{-i}}$, tels que $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$.

Soit x_i la variable de V telle que $\pi_i = \{x_i\}$. Plusieurs cas sont possibles :
– $Pa(x_i) = \varnothing$. D'après l'algorithme 1, nous avons $CPT(x_i) = x_i \succ \overline{x}_i$. Comme $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$, nous savons que $s_i = \overline{x}_i$. Pourtant, nous savons d'après l'algorithme 2 que $x_i \in s$. Nous avons une contradiction.
– $Pa(x_i) \neq \varnothing$. Nous savons d'après l'algorithme 1 que $CPT(x_i) = \{\bigvee_{v \in Pa(x_i)} v : \overline{x}_i \succ x_i\} \cup \{\bigwedge_{v \in Pa(x_i)} \overline{v} : x_i \succ \overline{x}_i\}$. Deux cas sont possibles :
  – Aucune variable de $Pa(x_i)$ n'est satisfaite : $\bigwedge_{v \in Pa(x_i)} \overline{v}$. Dans ce cas, nous savons que $x_i \succ \overline{x}_i$. Donc, comme $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$, nous avons $s_i = \overline{x}_i$. Mais nous savons d'après l'algorithme 2 que si $Pa(x_i) \subseteq Out$, c'est-à-dire si aucune variable de $Pa(x_i)$ n'est satisfaite, alors nous avons $x_i \in In$, et donc $x_i \in s$, ce qui amène une contradiction.
  – Au moins une variable de $Pa(x_i)$ est sa-

tisfaite : $\bigvee_{v \in Pa(x_i)} v$. Dans ce cas, nous savons que $\overline{x}_i \succ x_i$. Donc, comme $(s'_i, s_{-i}) \succ_i (s_i, s_{-i})$, $s_i = x_i$. Mais nous savons d'après l'algorithme 2 que si $Pa(x_i) \cap In \neq \varnothing$, c'est-à-dire si au moins une variable de $Pa(x_i)$ est satisfaite, alors nous avons $x_i \in Out$, et donc $\overline{x}_i \in s$, ce qui amène une contradiction.

Étudions à présent à la direction $\Leftarrow$.

Soit $s = (s_1, \ldots, s_n)$ un SPNE de G. Nous devons montrer que $s \in SP$, c'est-à-dire que $\forall i$, si $s_i = x_i$, alors $x_i \in In$, sinon $(s_i = \overline{x}_i)$ $x_i \in Out$.

- $Pa(x_i) = \varnothing$. Nous savons d'après l'algorithme 1 que nous avons $CPT(x_i) = x_i \succ \overline{x}_i$. Comme s est un SPNE, nous savons que $s_i = x_i$. De plus, nous savons d'après l'algorithme 2 que $x_i \in In$.
- $Pa(x_i) \neq \varnothing$. Nous savons d'après l'algorithme 1 que nous avons $CPT(x_i) = \{\bigvee_{v \in Pa(x_i)} v : \overline{x}_i \succ x_i\} \cup \{\bigwedge_{v \in Pa(x_i)} \overline{v} : x_i \succ \overline{x}_i\}$. Deux cas sont alors possibles :
  - Aucune variable de $Pa(x_i)$ n'est satisfaite. Nous avons alors $\bigwedge_{v \in Pa(x_i)} \overline{v}$ et nous savons dans ce cas que $x_i \succ \overline{x}_i$. Comme s est un SPNE, nous savons que, dans ce cas, $s_i = x_i$. Nous savons aussi d'après l'algorithme 3 que si $Pa(x_i) \subseteq Out$, c'est-à-dire si aucune variable de $Pa(x_i)$ n'est satisfaite, alors nous avons $x_i \in In$.
  - Au moins une variable de $Pa(x_i)$ est satisfaite. Nous avons alors $\bigvee_{v \in Pa(x_i)} v$ et nous savons dans ce cas que $\overline{x}_i \succ x_i$. Comme s est un SPNE, nous savons que $s_i = \overline{x}_i$. Nous savons également d'après l'algorithme 3 que si $Pa(x_i) \cap In \neq \varnothing$, c'est-à-dire si au moins une variable de $Pa(x_i)$ est satisfaite, alors nous avons $x_i \in Out$.

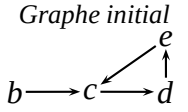
■

4.3 Gérer les circuits impairs

La suppression des circuits de longueur impaire a bien entendu une influence importance sur le calcul de(s) SPNE(s), et ce point peut être problématique si notre hypothèse initiale n'est pas acceptée : en général, les circuits de longueur impaire peuvent être considérés comme des paradoxes. Nous savons que certains circuits impairs ont un sens, en particulier lorsque ce ne sont pas des circuits impairs stricts[12]. Le travail présenté ici est préliminaire, et la suppression de ce genre de circuits permet de donner une traduction intéressante ayant des propriétés importantes. Nous sommes toutefois conscientes que le traitement des circuits impairs devra être fait (ce sera le sujet d'un prochain travail).

Exemple 7 *Soit* SA $= \langle \{b,c,d,e\}, \{(b,c), (c,d), (d,e), (e,c)\} \rangle$. *Le* SA *initial est cyclique et contient un circuit de longueur impaire qui sera supprimé. Le* SA *final ne contient que b.*

Graphe initial *Graphe final*



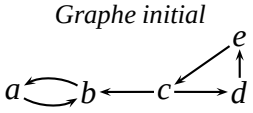
Nous obtenons en appliquant l'algorithme 1 $V = \{b\}$, $N = \{1\}$, *avec* $\pi_1 = \{b\}$ *et le CP-net suivant qui représente les préférences du joueur :*

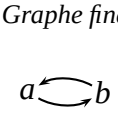
$$b \succ \overline{b} \quad B$$

G a donc un SPNE $\{b\}$ *qui correspond à l'extension préférée du* SA *final. Cependant, ce SPNE ne correspond pas à l'extension préférée du* SA *initial qui est l'ensemble* $\{b,d\}$. *Si nous supposons que les circuits de longueur impaire sont des paradoxes, et donc que leurs arguments ne sont pas significatifs, nous pouvons considérer que* $\{b\}$ *est une extension plus réaliste que* $\{b,d\}$[13].

Exemple 8 *Soit* SA $= \langle \{a,b,c,d,e\}, \{(a,b), (b,a), (c,b), (c,d), (d,e), (e,c)\} \rangle$. *Le* SA *initial est cyclique, et il contient un circuit impair qui sera supprimé. Le* SA *final ne contiendra que les variables a et b.*

Graphe initial *Graphe final*





Nous obtenons en appliquant l'algorithme 1 $V = \{a,b\}$, $N = \{1,2\}$, *avec* $\pi_1 = \{a\}$, $\pi_2 = \{b\}$ *et le CP-net suivant qui représente les préférences de tous les joueurs :*

$$\begin{array}{l} b : \overline{a} \succ a \\ \overline{b} : a \succ \overline{a} \end{array} \quad A \quad B \quad \begin{array}{l} a : \overline{b} \succ b \\ \overline{a} : b \succ \overline{b} \end{array}$$

[12] Par exemple quand on a en plus du circuit impair $a-b-c-a$, les attaques (b,a), (c,b) et (a,c). Dans ce cas, il existe en "surimpression" du circuit impair, trois circuits pairs simples ($a-b-a$, $b-c-b$, $a-c-a$), un autre circuit impair simple ($a-c-b-a$) et plusieurs circuits pairs non simples ($a-c-a-b-a$, ...).

[13] Notons toutefois que cette manière de traiter cet exemple ne correspond pas aux principales sémantiques pour l'acceptabilité.

G a donc deux SPNEs $\{a\overline{b}, \overline{a}b\}$ qui correspondent aux deux extensions préférées du SA *final. Cependant, le* SA *initial a une seule extension préférée : l'ensemble $\{a\}$. Prendre en compte dans ce cas l'extension $\{b\}$ signifie que l'attaque $c \rightarrow b$ est considérée comme n'étant pas significative (car un circuit impair est un paradoxe, donc ses arguments ne sont pas significatifs, et donc ils ne peuvent pas fournir une attaque réelle contre un autre argument).*

5 Travaux connexes

Les principaux travaux connexes nous concernant traitent des liens entre argumentation et jeux identifiés par Dung. Ce lien apparaît déjà dans [16] ; un SA y est utilisé pour résoudre un jeu coopératif classique (le problème du mariage stable) ; Dung utilise les arguments du SA pour représenter les issues possible du jeu, et la relation d'attaque pour exprimer les conflits entres les différentes issues. Ce lien a également été utilisé dans [8] afin de démontrer l'acceptabilité d'un argument *a*, et cela à partir d'un jeu dynamique dans lequel un joueur est pour l'acceptabilité de *a* et le second est contre. Les principales différences avec notre travail portent sur la nature du jeu (statique pour nous et dynamique ailleurs), puis sur le nombre et le rôle des joueurs.

Les seconds travaux connexes importants, qui ont déjà été mentionnés dans l'introduction, concernent le calcul des extensions préférées : des algorithmes ont déjà été proposés pour effectuer ce calcul (voir, par exemple [14, 17, 18, 9]). Et il est important de noter que nos algorithmes ne présentent pas de gain en efficacité par rapport aux algorithmes existants.

D'autres travaux connexes concernent l'utilisation d'un argument comme littéral dans une formule propositionnelle. Cette idée peut également être trouvée dans [13, 3, 12] (par exemple, dans [3], une caractérisation des extensions préférées est donnée sous la forme d'une formule propositionnelle).

Enfin, le dernier type de travaux connexes concerne le traitement des circuits de longueur impaire. Diverses approches existent dans la littérature : ces circuits peuvent apparaître dans un SA, mais sont interdits dans les extensions (voir par exemple [11]) ; ces circuits peuvent également être acceptés et traités comme des circuits de longueur paire pour le calcul des extensions (voir par exemple [2]). Notre approche correspond au premier cas : des circuits impairs peuvent apparaître dans un SA, mais ils seront supprimés pour le calcul des extensions (voir l'algorithme 1).

6 Conclusion

Nous avons montré comment traduire un système d'argumentation SA en un CP-jeu booléen, et comment ce jeu permet de calculer les extensions préférées du SA original en utilisant les équilibres de Nash en stratégies pures. Nous avons donné trois algorithmes formels permettant respectivement de transformer le SA en CP-jeu booléen, et de calculer les extensions préférées du SA. En outre, une fois les circuits impairs du SA éliminés, si le système d'argumentation SA' est acyclique, alors les extensions préférées du SA' sont calculables en temps polynomial.

Une limitation claire de nos résultats est que la traduction proposée élimine d'office les circuits de longueur impaire du système d'argumentation initial. Nous avons expliqué ce choix par le fait que notre travail est encore préliminaire, et que de tels systèmes d'argumentation ont d'importantes propriétés. Toutefois, il serait intéressant d'étudier des systèmes d'argumentation munis de circuits impairs, car certains de ces circuits peuvent avoir un sens. Une de nos pistes de travail est donc d'étudier si nos résultats s'appliquent toujours dans ce cas, et de modifier nos algorithmes en conséquence si ce n'est pas le cas.

Références

[1] Leila Amgoud, Nicolas Maudet, and Simon Parsons. Modelling dialogues using argumentation. In *Proceedings of ICMAS'00*, pages 31–38, 2000.

[2] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. SCC-recursiveness : a general schema for argumentation semantics. *Artifical Intelligence*, 168 :162–210, 2005.

[3] Philippe Besnard and Sylvie Doutre. Characterization of semantics for argument systems. In *Proceedings of KR'04*, pages 183–193, 2004.

[4] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach

to default reasoning. *Artificial Intelligence*, 93 :63–101, 1997.

[5] Elise Bonzon, Marie-Christine Lagasquie-Schiex, Jérôme Lang, and Bruno Zanuttini. Compact preference representation and Boolean games. *Journal of Autonomous Agents and Multi-Agent Systems*, 18(1) :1–35, 2009.

[6] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. CP-nets : A Tool for Representing and Reasoning with Conditional *Ceteris Paribus* Preference Statements. *Journal of Artificial Intelligence Research*, 21 :135–191, 2004.

[7] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. Preference-Based Constrained Optimization with CP-nets. *Computational Intelligence*, 20(2) :137–157, 2004.

[8] Claudette Cayrol, Sylvie Doutre, and Jérôme Mengin. Dialectical Proof Theories for the Credulous Preferred Semantics of Argumentation Frameworks. In *Proceedinfs of ECSQARU'01*, pages 668–679, 2001.

[9] Claudette Cayrol, Sylvie Doutre, and Jérôme Mengin. On Decision Problems Related to the Preferred Semantics for Argumentation Frameworks. *Journal of Logic and Computation*, 13(3) :377–403, 2003.

[10] Carlos I. Chesñevar, Ana G. Maguitman, and Ronald P. Loui. Logical models of argument. *ACM Computing surveys*, 32(4) :337–383, 2000.

[11] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Prudent semantics for argumentation frameworks. In *Proceedings of ICTAI'05*, pages 568–572, 2005.

[12] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Contrained argumentation frameworks. In *Proceedings of KR' 06*, pages 112–122, 2006.

[13] Nadia Creignou. The Class of Problems That are Linearly Equivalent to Satisfiability or a Uniform Method for Proving NP-Completeness. *Theorical Computer Sciences*, 145(1&2) :111–145, 1995.

[14] Sylvie Doutre, and Jérôme Mengin. Preferred Extensions of Argumentation Frameworks : Query Answering and Computation. In *Proceedings of IJCAR (LNAI 2083)*, pages 272-288, 2001.

[15] Sylvie Doutre. *Autour de la sémantique préférée des systèmes d'argumentation*. PhD thesis, Université Toulouse III, 2002.

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

[17] Paul E. Dunne and Trevor J.M. Bench-Capon. Complexity and combinatorial properties of argument systems. Technical report, University of Liverpool, Department of Computer Science (U.L.C.S.), 2001.

[18] Paul E. Dunne and Trevor J.M. Bench-Capon. Coherence in finite argument system. *Artificial Intelligence*, 141(1-2) :187–203, 2002.

[19] Paul Harrenstein. *Logic in Conflict*. PhD thesis, Utrecht University, 2004.

[20] Paul Harrenstein, Wiebe van der Hoek, John-Jules Meyer, and Cees Witteveen. Boolean Games. In *Proceedings of TARK'01*, pages 287–298, 2001.

[21] Paul Krause, Simon Ambler, Morten Elvang, and John Fox. A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11 (1) :113–131, 1995.

[22] Henry Prakken and Gerard Vreeswijk. Logics for defeasible argumentation. In *Handbook of Philosophical Logic*, volume 4, pages 218–319, 2002.