

# Recherche de chemins optimaux pour multiples agents

Ricou Jules

Mai 2020

## Résumé

Cette étude s'inscrit dans le cadre de mon stage de fin de Licence à l'Université d'Angers. Son but est la résolution du problème de recherche de chemins pour plusieurs agents sans qu'ils n'occupent une même position. L'encodage de ce sujet en un problème de satisfiabilité a pour objectif de minimiser le nombre d'étapes maximales pour que tous les agents aient atteint leur destination.

# Table des matières

<b>Introduction</b>	<b>3</b>
<b>1 Contexte</b>	<b>4</b>
1.1 Modélisation formelle du problème . . . . .	4
1.2 Problème de satisfiabilité . . . . .	5
<b>2 Traduction en problème de satisfiabilité</b>	<b>7</b>
2.1 Méthode de recherche de la solution optimale . . . . .	7
2.2 Expansion d'un graphe dans le temps . . . . .	7
2.3 Réduire l'espace de recherche . . . . .	9
2.4 Encodage en problème de satisfiabilité . . . . .	10
2.5 Conversion en forme normale conjonctive . . . . .	11
2.6 Taille des clauses et nombre de variables . . . . .	13
<b>3 Implémentation</b>	<b>14</b>
3.1 Format de fichier CPF . . . . .	14
3.2 Détermination de l'existence d'une solution . . . . .	14
3.3 Utilisation du solveur SAT <i>Glucose</i> . . . . .	15
3.4 Traduction du problème CPF en problème SAT . . . . .	15
3.5 Utilisation du programme . . . . .	15
3.6 Exemple d'exécution . . . . .	16
3.6.1 Utilisation de la réduction de l'espace de recherche . . . . .	16
3.6.2 Désactivation de la réduction de l'espace de recherche . . . . .	18
3.6.3 Comparaison de la taille de la formule avec et sans la réduction de l'espace de recherche . . . . .	20
<b>Conclusion</b>	<b>22</b>
<b>Références</b>	<b>23</b>
<b>Annexes</b>	<b>23</b>

## Introduction

La recherche de chemins pour multiples agents ou recherche coopérative de chemins consiste à déplacer plusieurs agents sans qu'ils n'entrent en collision. Chaque agent démarre à une position et doit en atteindre une autre. Plus précisément, chacun doit simultanément se déplacer sans être au même moment au même endroit. Ce problème vise à trouver une solution optimale. Sachant que le terme optimal aura, dans cette recherche, pour signification de minimiser la distance du chemin le plus long (1.1.2).

La résolution de ce problème est NP-difficile. La vérification d'une solution prend un temps polynomial mais trouver une solution se heurte à un nombre exponentiel de cas suivant le nombre d'agents et la taille de l'environnement.

Dans ce cas, il est sensé de convertir le problème originel en un problème de satisfiabilité pour bénéficier de la rapidité des solveurs SAT modernes.

Ce travail a pour vocation de modéliser ce problème grâce un encodage simple mais efficace [3] (2.4) et d'optimiser l'espace de recherche à l'aide d'un arbre décisionnel (2.3). Puis d'implémenter cet encodage et de le faire résoudre par un solveur SAT.

# 1 Contexte

Pour modéliser ce problème, l'environnement dans lequel évoluent les agents sera représenté par un graphe non orienté. Chaque agent est considéré comme une entité discrètes positionnée sur un unique noeud du graphe. Le temps sera modélisé par une séquence d'intervalle régulier ; à chaque interval, tout agent pourra se déplacer – ou non – vers un noeud voisin en suivant une arête du graphe. La contrainte de conflit impose qu'à aucun moment, deux agents ne se retrouvent sur un même noeud. Le but de ce problème est de trouver, pour chaque agent, un chemin de son noeud initial à son noeud final, en un temps minimal.

## 1.1 Modélisation formelle du problème

Soit  $G = (V, E)$ , un graphe non orienté modélisant l'environnement avec  $V = \{v_0, v_1, \dots, v_n\}$  un ensemble fini de noeuds et  $E \subseteq \{\{v_i, v_j\} | v_i, v_j \in V\}$  un ensemble d'arêtes entre deux noeuds du graphe. Ainsi que  $A = \{a_1, a_2, \dots, a_m\}$  un ensemble fini d'agents,  $m \leq n$ . Le temps, comme décrit plus tôt, est divisé en interval de temps discret  $t$ .

On note  $\alpha_t : A \rightarrow V$  un arrangement des agents à l'intervalle de temps  $t$ , de telle sorte que  $\alpha_t(a) \in V$  désigne le noeud qu'occupe l'agent  $a \in A$ . Un seul agent peut être positionné sur un noeud. Par conséquent,  $\alpha_t$  est uniquement inversible, et défini par  $\alpha_t^{-1}(v) \in A \cup \{\perp\}$ . Intuitivement,  $\alpha_t^{-1}$  désigne l'agent occupant le noeud  $v \in V$  à l'instant  $t$ , ou  $\perp$  si le noeud est vide.

### Définition 1.1.1 (Recherche coopérative de chemin):

Une instance d'un problème de recherche coopérative de chemin, ou CPF<sup>1</sup>, sera déclarée comme le quadruplé  $\Sigma = (G, A, \alpha_0, \alpha^+)$  ; avec  $\alpha_0$  et  $\alpha^+$ , la configuration de départ et, respectivement, de fin des agents. Une solution à cette instance est une séquence d'arrangements  $[\alpha_0, \alpha_1, \dots, \alpha_\tau]$  tel que  $\alpha_\tau = \alpha^+$  et  $\alpha_{t+1}$  est une transition valide depuis  $\alpha_t$  pour  $t = 0, 1, \dots, \tau - 1$ .

Une transition entre deux arrangements  $\alpha_i$  et  $\alpha_j$  est valide, si et seulement si, elle vérifie ces deux propriétés :

$$\forall a \in A, \alpha_i(a) = \alpha_j(a) \vee \{\alpha_i(a), \alpha_j(a)\} \in E \quad (1)$$

---

1. CPF : Cooperative pathfinding

**(Un agent ne se déplace pas ou suit une arête du graphe)**

$$\forall a, b \in A, a \neq b \implies \{\alpha_i(a), \alpha_j(a)\} \neq \{\alpha_i(b), \alpha_j(b)\} \quad (2)$$

**(Deux agents ne se croisent pas sur une même arête)**

**Définition 1.1.2** (Durée d'une solution):

La durée d'une solution est le temps nécessaire pour que tous les agents aient atteint leur destination. Pour une solution  $[\alpha_0, \alpha_1, \dots, \alpha_\tau]$ , sa durée est égale à  $\tau$ . Il est aussi possible de la calculer par  $\max_{a \in A} \xi(a)$  où  $\xi(a) \in \mathbb{N}$  est le nombre d'arête traversé par l'agent  $a$ .

## 1.2 Problème de satisfiabilité

Un problème SAT ou problème de satisfiabilité, est un problème de décision qui détermine s'il existe une assignation de variable rendant une formule de logique propositionnelle vraie.

**Définition 1.2.1** (Formule de logique propositionnelle):

Soit l'ensemble de valeurs booléennes  $\mathbb{B} = \{\perp, \top\}$ , et  $\mathbf{B}$  un ensemble infini et dénombrable de variables à valeurs dans  $\mathbb{B}$ . L'ensemble  $\mathbf{P}$  des formules de logique propositionnelle est construit comme suit :

- Si  $b \in \mathbb{B}$  alors  $b \in \mathbf{P}$
- Si  $b \in \mathbf{B}$  alors  $b \in \mathbf{P}$
- Si  $p \in \mathbf{P}$  alors  $\neg p \in \mathbf{P}$
- Si  $P, Q \in \mathbf{P}$  alors  $P \wedge Q \in \mathbf{P}$
- Si  $P, Q \in \mathbf{P}$  alors  $P \vee Q \in \mathbf{P}$
- Si  $P, Q \in \mathbf{P}$  alors  $P \implies Q \in \mathbf{P}$

**Définition 1.2.2** (Assignation):

Pour une formule  $P \in \mathbf{P}$  contenant les variables booléennes  $B \subset \mathbf{B}$ . Une assignation  $\Gamma : \mathbf{B} \rightarrow \mathbb{B}$  attribue une valeur booléenne de  $\mathbb{B}$  à chaque variable de la formule. On note  $\Gamma(P) \in \mathbf{P}$  la formule qui substitue à chaque variable  $b \in B$  de  $P$  à sa valeur booléenne  $\Gamma(b)$ .

**Exemple 1.2.2.1:**

Soit  $a, b, c \in \mathbf{B}$ , la formule  $P = a \wedge (b \vee \neg c)$ , et l'assignation  $\Gamma = \{(a \rightarrow \perp), (b \rightarrow \top), (c \rightarrow \perp)\}$ . Alors  $\Gamma(P) = \perp \wedge (\top \vee \neg \perp) = \perp$ .

**Définition 1.2.3** (Satisfiabilité d'une formule):

Une formule  $P \in \mathbf{P}$  est satisfiable s'il existe une assignation  $\Gamma$  tel que  $\Gamma(P) = \top$ .  
On notera alors  $\Gamma \models P$ .

## 2 Traduction en problème de satisfiabilité

Obtenir une solution optimale (du point de vue de la durée de la solution) est un problème NP, il peut donc être traduit en une instance d'un problème SAT comme le stipule le théorème de Cook. La formule propositionnelle  $F(\Sigma, \tau)$  est satisfiable si et seulement s'il existe une solution à  $\Sigma$  de durée maximale  $\tau$ .

### 2.1 Méthode de recherche de la solution optimale

Il existe plusieurs méthodes pour obtenir la solution optimale, la plus simple, et pourtant efficace, consiste à séquentiellement construire et résoudre la formule  $F(\Sigma, \tau)$  pour  $\tau = 0, 1, \dots$  jusqu'à ce que  $F(\Sigma, \tau)$  soit satisfiable (Pseudo-code de la stratégie est décrit dans 2.1.1). Il est à noter que cette méthode ne termine jamais s'il n'existe pas de solution à  $\Sigma$ . Pour éviter ce cas, il est possible d'utiliser un algorithme polynomial tel que PUSH-AND-SPIN [5] (3.2) vérifiant l'existence d'une solution.

---

**Algorithm 2.1.1** Trouve séquentiellement la solution avec la plus petite durée qui résout  $\Sigma$ . Si aucune solution n'est possible,  $\emptyset$  est retourné.

---

```
if  $\Sigma$  a une solution then
   $\tau \leftarrow 0$ 
  loop
     $F(\Sigma, \tau) \leftarrow \text{encode vers SAT}(\Sigma, \tau)$ 
    if  $\text{resoud}(F(\Sigma, \tau))$  then
       $s \leftarrow \text{extrait solution}(F(\Sigma, \tau))$ 
      return  $(s, \tau)$ 
    end if
  end loop
else
  return  $(\emptyset, \infty)$ 
end if
```

---

### 2.2 Expansion d'un graphe dans le temps

Un chemin dans le temps d'un agent sur le graphe  $G$  n'est pas nécessairement simple, il peut utiliser plusieurs fois un même noeud. Pour résoudre ce problème,

le graphe  $G$  sera étendu dans le temps. Chaque noeud aura  $\tau$  synonymes représentant la visite du noeud à l'instant  $t$ . Ainsi à chaque agent pourra être attribué un chemin simple dans le graphe étendu.

**Définition 2.2.1** (Expansion d'un graphe dans le temps  $TEG^\tau(G)$ ):

Soit  $G = (V, E)$  un graphe non-orienté et  $\tau \in \mathbb{N}$ , une expansion dans le temps du graphe  $G$  avec  $\tau + 1$  couche de temps (indexé de 0 à  $\tau$ ) est un graphe orienté  $TEG^\tau(G) = (V', E')$ ; avec :

$$\begin{aligned} V' &= \{v^t \mid v \in V \wedge t = 0, 1, \dots, \tau\} \\ E' &= \{(u^t, v^{t+1}), (v^t, u^{t+1}) \mid \{u, v\} \in E \wedge t = 0, 1, \dots, \tau - 1\} \cup \\ &\quad \{(v^t, v^{t+1}) \mid v \in V \wedge t = 0, 1, \dots, \tau - 1\}. \end{aligned}$$

La recherche coopérative de chemin de durée  $\tau$  peut être considéré comme une recherche d'un ensemble de chemins disjoints et sans croisement dans le graphe étendu dans le temps.

**Définition 2.2.2** (Ensemble de chemins disjoints et sans croisement):

Soit l'ensemble de chemins  $\Pi = \{\pi_0, \pi_1, \dots, \pi_m\}$  dans  $TEG^\tau(G) = (V', E')$ , tel que pour tout  $i \leq m$ ,  $\pi_i$  connecte  $v^0 \in V'$  à  $u^\tau \in V'$ ,  $v, u \in V$ , c'est-à-dire,  $\pi_i$  est une séquence jointe d'arêtes dans  $E'$ . Cet ensemble  $\Pi$  est dit disjoint et sans croisement si et seulement si pour toutes paires de chemins  $\pi_i, \pi_j$ ,  $i \neq j$  :

$$\begin{aligned} V(\pi_i) \cap V(\pi_j) &= \emptyset \\ \text{avec } V(\pi_k) &\subseteq V', \text{ l'ensemble des noeuds parcourus de } \pi_k \end{aligned} \quad (3)$$

**(Les noeuds parcourus annotés du temps)**

$$\begin{aligned} TE(\pi_i) \cap TE(\pi_j) &= \emptyset \\ \text{avec } TE(\pi_k) &= \{\{v_i, v_j\}_{t+1}^t \mid (v_i^t, v_j^{t+1}) \in \pi_k\} \end{aligned} \quad (4)$$

**(Les arêtes (non orientées) parcourus annotées du temps)**

La propriété (3) assure que deux chemins de  $\Pi$  n'occupent un même noeud. Et (4) garanti qu'aucun croisement n'est possible.

**Proposition 2.2.3:**

Une solution à un problème CPF  $\Sigma = (G, A, \alpha_0, \alpha^+)$  avec  $A = \{a_1, a_2, \dots, a_m\}$  et



de durée  $\tau$ , existe si et seulement s'il existe un ensemble  $\Pi = \{\pi_0, \pi_1, \dots, \pi_m\}$  de chemins disjoints et sans croisement dans  $TEG^\tau(G)$  tel que  $\pi_i$  connecte  $\alpha_0(a_i)^0$  à  $\alpha^+(a_i)^\tau$  pour  $i = 1, 2, \dots, m$ .

*Démonstration.* Soit un problème CPF  $\Sigma = (G, A, \alpha_0, \alpha^+)$  et une de ses solutions  $\mathcal{S} = [\alpha_0, \alpha_1, \dots, \alpha_\tau]$ , avec  $A = \{a_1, a_2, \dots, a_m\}$  et  $G = (V, E)$ .

L'ensemble des chemins  $\Pi = \{\pi_0, \pi_1, \dots, \pi_m\}$  de  $TEG^\tau(G)$  disjoints et sans croisement peut être construit à partir de  $\mathcal{S}$  en définissant le chemin  $\pi_i$  comme la trajectoire de l'agent  $a_i \in A$ . Plus particulièrement :

$$\pi_i = [(\alpha_0(a_i)^0, \alpha_1(a_i)^1), (\alpha_1(a_i)^1, \alpha_2(a_i)^2), \dots, (\alpha_{\tau-1}(a_i)^{\tau-1}, \alpha_\tau(a_i)^\tau)]$$

De plus,  $\pi_i$  est un chemin de  $TEG^\tau(G)$  car, étant donné que  $\{\alpha_t(a_i), \alpha_{t+1}(a_i)\} \in E$ ,  $(\alpha_t(a_i)^t, \alpha_{t+1}(a_i)^{t+1}) \in E'$  par construction de  $TEG^\tau(G) = (V', E')$ . Evidemment,  $\pi_i$  connecte  $\alpha_0(a_i)^0$  à  $\alpha_\tau(a_i)^\tau$ . Il ne reste plus qu'à vérifier que les chemins soient disjoints et sans croisement. On sait que  $V(\pi_i)$  est l'ensemble  $\{\alpha_0(a_i)^0, \alpha_1(a_i)^1, \dots, \alpha_\tau(a_i)^\tau\}$ , or, un arrangement ne peut affecter deux agents sur un même noeud, par conséquent (3) est respecté. Aucun croisement ne peut avoir lieu (4) sinon la transition entre deux arrangements (2) ne serait pas respectée.

Montrons maintenant qu'il est possible de construire une solution du problème  $\Sigma = (G = (V, E), A, \alpha_0, \alpha^+)$  de durée  $\tau$  à partir d'un ensemble de chemins  $\Pi = \{\pi_0, \pi_1, \dots, \pi_m\}$  de  $TEG^\tau(G)$  disjoints et sans croisement où chaque chemin  $\pi_i$  connecte  $\alpha_0(a_i)$  à  $\alpha_+(a_i)$ .

Assumons l'existence d'un tel ensemble. On définit alors  $\alpha_t(a_i) = v_l$  avec  $\pi_i = [(v_0^0, v_1^1), (v_1^1, v_2^2), \dots, (v_{p-1}^{\tau-1}, v_p^\tau)]$ ,  $v_l \in V$ ,  $l = 0, 1, \dots, p$ ,  $t = 0, 1, \dots, \tau$ . Les chemins sont disjoints, donc les arrangements sont bien formés. De plus, par construction de  $TEG^\tau(G) = (V', E')$ , pour toute arête orientée  $(v^t, u^{t+1}) \in \pi_i$  implique que  $v = u$  ou  $\{v, u\} \in E$  (1). Enfin aucun chemin de  $\Pi$  ne se croise (4), impliquant nécessairement que toutes transitions entre deux arrangements soient valides (2).

□

## 2.3 Réduire l'espace de recherche

Cette section emprunte l'idée d'un MDD<sup>2</sup> de l'algorithme ICTS [1] pour réduire l'espace de recherche. Cela permet d'améliorer considérablement la vitesse

---

2. MDD : Multi-value decision diagram

de résolution du problème (2.6) et de réduire le nombre de variables et la taille des clauses. Intuitivement, le MDD d'un agent est le graphe n'incluant que les noeuds accessibles dans le temps imparti.

**Définition 2.3.1** (Reduction de l'espace de recherche d'un agent  $MDD_i^\tau$ ):

Soit le problème CPF  $\Sigma = (G, A, \alpha_0, \alpha^+)$  avec  $G = \langle V, E \rangle$ ,  $A = \{a_1, a_2, \dots, a_m\}$ . Lors de la recherche d'une solution de durée  $\tau$ , on notera  $MDD_i^\tau \subseteq TEG^\tau = (V', E')$  l'environnement de l'agent  $a_i \in A$ ,  $i = 1, 2, \dots, m$ . Plus précisément, sachant que  $\xi(v, u)$  est le coût de déplacement minimal entre deux noeuds  $v, u \in V$ ,  $MDD_i^\tau = (V^-, E^-)$  avec :

$$\begin{aligned} V^- &= \{v^t \mid \xi(\alpha_0(a_i), v) + \xi(v, \alpha^+(a_i)) \leq \tau, v^t \in V'\} \\ E^- &= \{(v, u) \mid (v, u) \in E', v, u \in V^-\} \end{aligned}$$

**Proposition 2.3.2:**

S'il existe une solution de durée  $\tau$  pour un problème CPF  $\Sigma = (G, A, \alpha_0, \alpha^+)$ , avec  $G = (V, E)$  et donc un ensemble de chemins dans  $TEG^\tau(G)$  disjoints et sans croisement  $\Pi = \{\pi_0, \pi_1, \dots, \pi_m\}$  alors tout chemin  $\pi_i \in \Pi$  est dans  $MDD_i^\tau$ .

*Démonstration.* Supposons l'existence d'un chemin  $\pi_i$  de l'ensemble  $\Pi$  qui n'est pas contenu dans  $MDD_i^\tau$ . Alors il existe un  $v \in V(\pi_i)$  tel que  $\xi(\alpha_0(a_i), v) + \xi(v, \alpha^+(a_i)) > \tau$ ,  $a_i \in A$ . Or  $\pi_i$  est un chemin dans  $TEG^\tau(G)$ , c'est-à-dire de longueur  $\tau$ . Donc tous chemins  $\pi_i \in \Pi$  est un chemin dans  $MDD_i^\tau$ .  $\square$

## 2.4 Encodage en problème de satisfiabilité

Pour transformer le problème de recherche de chemins, le concept de graphe étendu dans le temps est très important car il permet de représenter tous les arrangements possibles des agents à tout temps. L'encodage utilisé s'inspire de l'encodage DIRECT [2], à l'exception que nos prémisses diffèrent légèrement : notre modèle autorise qu'un agent entre dans un noeud occupé si celui-ci contient un agent qui en sort dans le même interval de temps.

**Définition 2.4.1** (Encodage DIRECT  $F(\Sigma, \tau)$ ):

Soit  $\Sigma = (G, A, \alpha_0, \alpha_+)$  une instance d'un problème CPF avec  $G = (V, E)$  et  $\tau \in \mathbb{N}$  la durée maximale, l'expansion de ce graphe, et sa réduction pour chaque agent  $a_i \in A$  nous donne  $MDD_i^\tau(G) = (V', E')$ . Pour encoder ce problème, on

défini les variables  $\chi_{a,v}^t$  comme représentant la présence de l'agent  $a \in A$  sur le noeud  $v^t \in V'$ . Les contraintes suivantes assurent la validité du modèle défini :

$$\chi_{a,v}^t \implies \bigvee_{u \mid (v^t, u^{t+1}) \in E'} \chi_{a,u}^{t+1} \quad (5)$$

$$\forall a \in A, \forall v \in V, \forall t \in \{0, 1, \dots, \tau - 1\}$$

**(Un agent suit une arête du graphe)**

$$\chi_{a,v}^t \implies \bigwedge_{u \in V, u \neq v} \neg \chi_{a,u}^t \quad (6)$$

$$\forall a \in A, \forall v \in V, \forall t \in \{0, 1, \dots, \tau\}$$

**(Un agent est positionné sur un noeud au maximum par intervalle de temps)**

$$\chi_{a,v}^t \implies \bigwedge_{b \in A, a \neq b} \neg \chi_{b,v}^t \quad (7)$$

$$\forall a \in A, \forall v \in V, \forall t \in \{0, 1, \dots, \tau\}$$

**(Un noeud contient un agent au maximum par intervalle de temps)**

$$\chi_{a,v}^t \wedge \chi_{a,u}^{t+1} \implies \bigwedge_{b \in A, a \neq b} \neg (\chi_{b,u}^t \wedge \chi_{b,v}^{t+1}) \quad (8)$$

$$\forall a \in A, \forall \{v, u\} \in E, \forall t \in \{0, 1, \dots, \tau - 1\}$$

**(Deux agents ne se croisent pas)**

## 2.5 Conversion en forme normale conjonctive

Afin d'utiliser un solveur SAT, les contraintes doivent être en forme normale conjonctive.

**Définition 2.5.1** (Forme normale conjonctive):

Une formule propositionnelle est en forme normale conjonctive lorsque celle-ci est une conjonction de clauses; où chaque clause est une disjonction de littéraux. Un littéral est une variable ou une valeur booléenne, de plus la négation d'un littéral est également un littéral.

**Exemple 2.5.1.1:**

Voici une liste non exhaustive de formules en forme normale conjonctive :  $A \wedge (B \vee \neg C)$ ,  $A$ ,  $(A \vee \neg B) \wedge (\neg A \vee C)$ ... ; ou d'autres qui ne respectent pas cette propriété  $\neg(A \vee B)$ ,  $A \vee (B \wedge \neg C)$ ...

Transformer les contraintes précédentes (2.4.1) en forme normale conjonctive est trivial grâce aux lois de De Morgan et de distributivité. Une implication  $A \implies B$  pourra aussi être éliminée et remplacée par  $\neg A \vee B$ .

$$\begin{aligned}
\chi_{a,v}^t &\implies \bigvee_{u \mid (v^t, u^{t+1}) \in E'} \chi_{a,u}^{t+1} \\
\iff \neg \chi_{a,v}^t &\vee \bigvee_{u \mid (v^t, u^{t+1}) \in E'} \chi_{a,u}^{t+1} \quad (9) \\
&\forall a \in A, \forall v \in V, \forall t \in \{0, 1, \dots, \tau - 1\}
\end{aligned}$$

**(Un agent suit une arête du graphe)**

$$\begin{aligned}
\chi_{a,v}^t &\implies \bigwedge_{u \in V, u \neq v} \neg \chi_{a,u}^t \\
\iff \bigwedge_{u \in V, u \neq v} &(\neg \chi_{a,v}^t \vee \neg \chi_{a,u}^t) \quad (10) \\
&\forall a \in A, \forall v \in V, \forall t \in \{0, 1, \dots, \tau\}
\end{aligned}$$

**(Un agent est positionné sur un noeud au maximum par intervalle de temps)**

$$\begin{aligned}
\chi_{a,v}^t &\implies \bigwedge_{b \in A, a \neq b} \neg \chi_{b,v}^t \\
\iff \bigwedge_{b \in A, a \neq b} &(\neg \chi_{a,v}^t \vee \neg \chi_{b,v}^t) \quad (11) \\
&\forall a \in A, \forall v \in V, \forall t \in \{0, 1, \dots, \tau\}
\end{aligned}$$

**(Un noeud contient un agent au maximum par intervalle de temps)**

$$\begin{aligned}
\chi_{a,v}^t \wedge \chi_{a,u}^{t+1} &\implies \bigwedge_{b \in A, a \neq b} \neg(\chi_{b,u}^t \wedge \chi_{b,v}^{t+1}) \\
\iff \bigwedge_{b \in A, a \neq b} &\neg\chi_{a,v}^t \vee \neg\chi_{a,u}^{t+1} \vee \neg\chi_{b,u}^t \vee \neg\chi_{b,v}^{t+1} \quad (12) \\
&\forall a \in A, \forall \{v, u\} \in E, \forall t \in \{0, 1, \dots, \tau - 1\}
\end{aligned}$$

**(Deux agents ne se croisent pas)**

## 2.6 Taille des clauses et nombre de variables

La modélisation du problème CPF a pour but d'être résolu par un solveur SAT, ainsi pour accélérer ce processus, prendre en compte le fonctionnement des solveurs SAT est primordial. L'un des facteurs les plus importants est la taille de la formule, le nombre de clauses et de variables doit être minimal. Même si ce travail n'a pas pour but de comparer d'autres modélisations existantes, il reste important de calculer la taille de la formule générée.

Pour une formule  $F(\Sigma, \tau)$  d'une instance  $\Sigma = (G, A, \alpha_0, \alpha_+)$  sur le graphe  $G = (V, E)$ , le nombre de variables est simple à calculer :  $O(\tau \cdot |A| \cdot |V|)$ . Ensuite chaque contraintes ajoutent de clauses. La propriété de déplacement 9 contient  $O(\tau \cdot |A| \cdot |V|)$  clauses. La propriété sur l'unicité d'une position par agent et par intervalle 10 ajoute  $O(\tau \cdot |A|^2 \cdot |V|)$  clauses. La propriété sur l'unicité d'un agent par noeud par intervalle 11 ajoute  $O(\tau \cdot |A| \cdot |V|^2)$  clauses. Et enfin, la contrainte qui impose l'absence de croisement (12) ajoute  $O(\tau \cdot |A|^2 \cdot |V|)$  clauses.

La réduction de l'espace de recherche entraîne une diminution de la taille de la formule. La topologie du graphe influence son impacte. C'est pour cette raison qu'une comparaison sera faites sur des exemples concrets (3.6.3).

## 3 Implémentation

Maintenant que le problème a été modélisé, la mise en pratique consiste à développer un programme qui traduit un problème CPF en un problème SAT et de le résoudre. La première étape est résolue par l'introduction d'un nouveau format de fichier (3.1). La résolution du problème SAT est effectuée par un programme externe *Glucose* (3.3). Et la traduction en problème SAT sera faite simplement en suivant les formes normales conjonctives précédemment définies (2.5).

### 3.1 Format de fichier CPF

Afin de définir le graphe, l'arrangement initial et l'arrangement final, un nouveau format de fichier est défini. Celui-ci est simple car le graphe n'est pas directionnel et les arêtes n'ont aucun poids. Pour plus de flexibilité, toute ligne commençant par "#" est ignorée, servant ainsi de commentaire. Autrement, le format est constitué d'une série de nombres séparés par un espace. En voici un exemple :

```
# Number of nodes
5

# Number of edges
4

# Graph's edges
0 1
1 2
1 4
2 3

# Number of agents
2

# Agents initial and goal nodes
0 2
3 4
```

### 3.2 Détermination de l'existence d'une solution

Pour déterminer si un problème CPF a une solution, la méthode de recherche itérative (2.1) ne fonctionne pas ; dans le cas où aucune solution n'existe, cette

méthode ne termine jamais. Pour compenser ce défaut, un algorithme supplémentaire tel que PUSH-AND-SPIN [5] peut-être ajouté en début de programme. Cependant, son implémentation s'est montré trop complexe pour être dans la version finale du programme.

### 3.3 Utilisation du solveur SAT *Glucose*

Pour résoudre le problème SAT, l'utilisation d'un solveur tel que *Glucose* est primordial au vu de la complexité de ce programme. Tel quel, *Glucose* utilise un fichier d'entrée au format *dimacs*. Afin d'éviter cet intermédiaire, *Glucose* a été légèrement modifié afin de pouvoir injecter directement les clauses. Cela permet aussi d'éviter une utilisation de *Glucose* à chaque test de la méthode itérative (2.1).

### 3.4 Traduction du problème CPF en problème SAT

La création de la formule est triviale, une variable est créée pour chaque noeud du *TEG* et les clauses sont ajoutées en suivant leurs définitions (2.5).

La réduction de l'espace de recherche complexifie quelque peu le programme. Pour déterminer les variables nécessairement fausses, deux parcours en largeur sont utilisés à partir du noeud initial et final de chaque agent. En effet, le parcours en profondeur permet non seulement d'être calculé itérativement mais aussi d'éviter le parcours du graphe dans sa totalité.

Pour déterminer si une variable  $\chi_{a,v}^t$  existe pour une durée maximale  $\tau$ , alors  $t \geq \xi(\alpha_0(a), v)$  et  $\tau - t \geq \xi(v, \alpha_+(a))$ ,  $\xi(v, u)$  étant égale à la plus petite distance entre  $v$  et  $u$ ; ou, dans notre cas, la profondeur de  $u$  par rapport à  $v$ . Lors de la création d'une clause contenant une variable fausse, celle-ci est omise si elle contient la négation de cette variable. Autrement le terme contenant la variable est supprimé de la clause.

### 3.5 Utilisation du programme

Le solveur contient multiples arguments, tous décrits avec l'option `--help` comme présenté ci-dessous. Le programme utilise le format CPF (3.1) pour définir le problème CPF à résoudre. Une option importante est `--no-mdd` pour désactiver l'optimisation de réduction de l'espace de recherche dans un but de comparaison.

```

Usage: ./build/solver <options> --input=<file>
       --input=<file>           File in CPF format [REQUIRED]
Options:
  --min-makespan=<value> Minimum makespan researched
  --max-makespan=<value> Maximum makespan researched
  --max-time=<value>      Maximum amount of seconds to
                        solve the CPF
  --trust                  Don't verify that a solution
                        exists (Doesn't do anything)
  --no-mdd                 Don't reduce search space
  --output=<file>          Write path of all agents to <file>
                        >, each line is a path, each path is a sequence of
                        number representing nodes

```

## 3.6 Exemple d'exécution

Les deux exemples d'exécution dans les sous-sections suivantes ont été établis sur le problème défini dans le fichier `impl/test/grid_test.cpf`<sup>3</sup>. Celui-ci a été généré<sup>4</sup> à partir d'une grille de taille 10 par 10 ; où chaque cellule a une probabilité de 10% d'être un obstacle non traversable et 20% d'être la cellule initiale d'un agent. À chaque agent a été assigné une destination aléatoire parmi les cellules sans obstacle.

### 3.6.1 Utilisation de la réduction de l'espace de recherche

Voici les résultats obtenus utilisant le programme avec les options : `./solver --output=result_with_mdd.res_cpf --input=test/grid_test.cpf`. On peut remarquer que l'utilisation du MDD permet d'éliminer les itérations avec une durée non suffisante pour que tous les agents aient au moins un chemin.

```

Generating SAT problem with a bounded makespan of 0...
  No path for agent 0 found in the MDD
  Took 0.931496ms
Generating SAT problem with a bounded makespan of 1...
  No path for agent 0 found in the MDD
  Took 1.94534ms
Generating SAT problem with a bounded makespan of 2...
  No path for agent 0 found in the MDD
  Took 4.46273ms
Generating SAT problem with a bounded makespan of 3...

```

3. Voir annexe Utilisation du vérificateur/afficheur de solution

4. Voir annexe Utilisation du générateur de problème CPF



```

        No path for agent 0 found in the MDD
        Took 10.5582ms
Generating SAT problem with a bounded makespan of 4...
        No path for agent 0 found in the MDD
        Took 14.6095ms
Generating SAT problem with a bounded makespan of 5...
        No path for agent 0 found in the MDD
        Took 23.7924ms
Generating SAT problem with a bounded makespan of 6...
        No path for agent 0 found in the MDD
        Took 32.186ms
Generating SAT problem with a bounded makespan of 7...
        No path for agent 3 found in the MDD
        Took 47.032ms
Generating SAT problem with a bounded makespan of 8...
        No path for agent 3 found in the MDD
        Took 66.0722ms
Generating SAT problem with a bounded makespan of 9...
        No path for agent 3 found in the MDD
        Took 93.3454ms
Generating SAT problem with a bounded makespan of 10...
        No path for agent 3 found in the MDD
        Took 135.354ms
Generating SAT problem with a bounded makespan of 11...
        #Variables: 1102
        #Clauses: 22176
        Solving...
        Took 1709.73ms
        Successfully solved
Total time: 2140.22ms
Path of all agents:
    Agent #0: #56, #66, #65, #64, #64, #74, #84, #83, #93,
              #93, #83, #82,
    Agent #1: #91, #81, #82, #83, #84, #85, #86, #85, #75,
              #85, #84, #74,
    Agent #2: #36, #26, #26, #16, #15, #14, #15, #16, #6,
              #7, #6, #5,
    Agent #3: #9, #19, #18, #17, #16, #26, #36, #35, #45,
              #44, #43, #42,
    Agent #4: #15, #14, #24, #34, #44, #54, #64, #65, #66,
              #67, #77, #78,
    Agent #5: #48, #38, #28, #18, #8, #7, #6, #5, #15, #5,
              #5, #4,
    Agent #6: #18, #17, #16, #15, #14, #24, #23, #22, #12,
              #11, #12, #22,

```

```
Agent #7: #46, #56, #66, #76, #77, #78, #79, #69, #69,  
         #69, #69, #79,  
Agent #8: #85, #86, #96, #97, #97, #97, #97, #96, #86,  
         #76, #66, #76,  
Agent #9: #8, #9, #9, #19, #29, #28, #18, #17, #16, #15,  
         #14, #24,  
Writing to 'result_with_mdd_.res_cpf'... Done
```

### 3.6.2 Désactivation de la réduction de l'espace de recherche

Voici les résultats obtenus utilisant le programme avec les options : `./ solver  
--output=result_without_mdd.res_cpf --input=test/grid_test.cpf --no-mdd.`

```
Generating SAT problem with a bounded makespan of 0...  
#Variables: 1000  
#Clauses: 110000  
Solving...  
Took 356.587ms  
Failed to solve.  
Generating SAT problem with a bounded makespan of 1...  
#Variables: 2000  
#Clauses: 239880  
Solving...  
Took 982.789ms  
Failed to solve.  
Generating SAT problem with a bounded makespan of 2...  
#Variables: 3000  
#Clauses: 369760  
Solving...  
Took 1834.17ms  
Failed to solve.  
Generating SAT problem with a bounded makespan of 3...  
#Variables: 4000  
#Clauses: 499640  
Solving...  
Took 2271.56ms  
Failed to solve.  
Generating SAT problem with a bounded makespan of 4...  
#Variables: 5000  
#Clauses: 629520  
Solving...  
Took 2869.91ms  
Failed to solve.  
Generating SAT problem with a bounded makespan of 5...  
#Variables: 6000
```

```

    #Clauses: 759400
    Solving ...
    Took 3493.55ms
    Failed to solve.
Generating SAT problem with a bounded makespan of 6...
    #Variables: 7000
    #Clauses: 889280
    Solving ...
    Took 4032.81ms
    Failed to solve.
Generating SAT problem with a bounded makespan of 7...
    #Variables: 8000
    #Clauses: 1019160
    Solving ...
    Took 4874.62ms
    Failed to solve.
Generating SAT problem with a bounded makespan of 8...
    #Variables: 9000
    #Clauses: 1149040
    Solving ...
    Took 5628.24ms
    Failed to solve.
Generating SAT problem with a bounded makespan of 9...
    #Variables: 10000
    #Clauses: 1278920
    Solving ...
    Took 6395.6ms
    Failed to solve.
Generating SAT problem with a bounded makespan of 10...
    #Variables: 11000
    #Clauses: 1408800
    Solving ...
    Took 6791.73ms
    Failed to solve.
Generating SAT problem with a bounded makespan of 11...
    #Variables: 12000
    #Clauses: 1538680
    Solving ...
    Took 8409.91ms
    Successfully solved
Total time: 47941.8ms
Path of all agents:
    Agent #0: #56, #66, #76, #75, #74, #73, #73, #83, #82,
               #81, #82, #82,

```

```

Agent #1: #91, #81, #82, #83, #83, #82, #83, #84, #84,
          #94, #84, #74,
Agent #2: #36, #26, #26, #16, #15, #14, #15, #16, #6,
          #16, #15, #5,
Agent #3: #9, #8, #7, #6, #16, #26, #36, #35, #45, #44,
          #43, #42,
Agent #4: #15, #14, #24, #34, #44, #54, #64, #65, #66,
          #67, #77, #78,
Agent #5: #48, #48, #38, #28, #18, #8, #7, #6, #5, #5,
          #4, #4,
Agent #6: #18, #17, #16, #15, #14, #24, #23, #22, #21,
          #11, #12, #22,
Agent #7: #46, #56, #57, #67, #77, #78, #79, #69, #69,
          #69, #69, #79,
Agent #8: #85, #86, #87, #97, #97, #97, #97, #96, #86,
          #76, #66, #76,
Agent #9: #8, #7, #6, #5, #4, #3, #4, #14, #15, #15,
          #14, #24,
Writing to 'result_with_mdd_.res_cpf'... Done

```

### 3.6.3 Comparaison de la taille de la formule avec et sans la réduction de l'espace de recherche

Le tableau suivant compare la résolution avec et sans la réduction de l'espace de recherche. Voici les exemples utilisés :

- *Grand environnement* : grille 12x12, 15 obstacles et 70 agents
- *Beaucoup d'obstacles* : grille 6x6, 4 obstacles et 28 agents
- *Beaucoup d'agents* : grille 6x6, 15 obstacles et 4 agents

	Avec MDD			Sans MDD		
	#Var	#Clauses	Temps	#Var	#Clauses	Temps
Grand environnement	66k	13.4M	83s	240k	100M	4100s
Beaucoup d'obstacles	1689	53177	382ms	9072	11.7M	1970ms
Beaucoup d'agents	53	208	1.4ms	756	30k	56ms

L'utilisation d'un MDD améliore grandement l'efficacité de l'algorithme en diminuant la taille de la formule par plusieurs ordres de grandeurs. Même si, pour certains problèmes, lorsque la durée maximale permet à tous les agents d'atteindre tous les noeuds du graphe, la taille de la formule est la même ; le temps utilisé pour construire les MDDs est en partie compensé par les itérations précédentes prenant partie de cette optimisation. On peut remarquer que

la vérification que tout agent possède au minimum un chemin pour la durée demandée est un biproduct trivial lorsque le MDD est généré ; éliminant totalement l'utilisation de solveur SAT pour cette itération.

## Conclusion

La formalisation d'un problème de recherche coopérative de chemins permet une compréhension détaillée du problème et une conversion en un problème de satisfiabilité sans souci. L'utilisation d'un graphe étendu dans le temps est essentielle pour générer une formule logique résolvant un problème CPF de durée maximale. En augmentant la durée allouée, une solution optimale est obtainable.

L'utilisation d'un solveur SAT moderne tel que *Glucose* a permis d'accéder à de multiples optimisations bénéfique à la recherche d'une solution au problème. Le choix de l'encodage DIRECT a non seulement permis une modélisation simplifiée, comparé à d'autres méthodes [3], mais aussi une implémentation rapide et sans encombre. On a aussi remarqué que la réduction de l'espace de recherche par un MDD a contribué significativement à l'efficacité du programme.

Finalement, peu de problèmes se sont imposés. Cependant, par manque de temps, l'algorithme PUSH-AND-SPIN [5] n'a pas pu faire partie de l'implémentation finale. Même s'il n'est pas le point central du sujet, son ajout aurait sans aucun doute rendu ce programme plus simple d'utilisation.

Pour conclure, ce travail s'est engagé à définir une méthode pour obtenir une solution optimale. Seulement, minimiser la longueur du chemin le plus long a pour conséquence de ne pas contraindre la longueur des autres chemins de cette solution. Par exemple, une solution contenant deux chemins de longueur  $n$  n'est pas moins optimale qu'une solution avec un chemin de longueur  $n$  et un autre vide. Pour résoudre ce problème, il est alors possible d'encoder une borne supérieure à la somme des longueurs des chemins [4] en plus de la borne supérieure sur la durée de la solution.

Une optimisation supplémentaire peut-être aussi implémenté, en effet sur certaines instances d'un problème CPF, les agents peuvent être séparés en groupes indépendants, dans le but de réduire le nombre de clauses et leurs tailles. Pour sa réalisation, une division du graphe doit être présente afin qu'aucun agent d'une des deux parties ait pour destination un noeud de l'autre partie du graphe. De plus la division du graphe ne doit pas empêcher toutes les solutions optimales. Sans un travail plus approfondi du sujet, cette dernière propriété est complexe à définir mais toutefois possible.

## Références

- [1] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195 :662–667, 01 2011.
- [2] P. Surynek. A simple approach to solving cooperative path-finding as propositional satisfiability works well. pages 827–833, 12 2014.
- [3] P. Surynek. Makespan optimal solving of cooperative path-finding via reductions to propositional satisfiability. 10 2016.
- [4] P. Surynek. Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. *Annals of Mathematics and Artificial Intelligence*, 81, 08 2017.
- [5] E. Turki. A complete multi-robot path-planning algorithm. *Autonomous Agents and Multi-Agent Systems*, 32, 06 2018.

## Annexes

### Utilisation du générateur de problème CPF

Ce programme permet de générer un problème CPF au format du même nom. L'instance est constitué d'une grille de noeuds reliés a leurs voisins, sauf en cas d'obstacle. Les agents et les obstacles sont affecté aléatoirement dans la grille.

Usage: `./build/generator <options>`

Options:

```
—output=<file>  File writen to, in CPF format ,
                  otherwise write to standard output stream
—size=<value>    Size of the grid 's size [DEFAULT: 4]
—wall%=<value>   Percentage of walls in the grid [
                  DEFAULT: 20]
—agent%=<value>  Percentage of agents in the grid [
                  DEFAULT: 10]
—display=<file>  Display the grid in the file , no value
                  will write to standard output stream
```

### Utilisation du vérificateur/afficheur de solution

Ce programme annexe permet de vérifier le bon fonctionnement du programme principale. Si le graphe est une grille comme produit par le générateur (Voir

annexe Utilisation du générateur de problème CPF) alors la solution est apte à être affiché dans la console.

```
Parameter —graph required
Usage: ./build/verifier <options> —graph=<file.cpf> —result=<
file.res_cpf>
    —graph=<file>    File in CPF format describing the graph
                    and the agents [REQUIRED]
    —result=<file>    File outputted by the solver when
                    solving the problem in —graph [REQUIRED]
Options:
    —show-as-grid    Print the successive time steps, the
                    graph *must be* a squared grid, like the problems
                    produced by the generator
```