Contrôle des feux de circulation distribué sur plusieurs intersections

Gestion d'une zone urbaine ciblée

Au chapitre précédent, nous avons proposé une méthode qui permet de déployer des capteurs sur une large zone urbaine. Dans ce chapitre, nous nous intéressons aux applications qu'il est possible d'exécuter par dessus ce type de réseau. Si les systèmes de gestion sont nombreux, à commencer par ceux s'occupant du stationnement, nous nous tournons plutôt du côté de la gestion des feux de circulation, dont les enjeux – introduits notamment au chapitre 2 – sont cruciaux.

Outre les temps de déplacement, la perte de temps dans la circulation augmente la pollution et le bruit, ce qui en fait un élément clé en zone urbaine. De plus, les embouteillages ont tendance à s'étendre depuis leur point de formation. Une gestion adaptative des feux de circulation, à l'aide d'un système distribué tel que nous venons de le décrire, pourrait empêcher cette expansion et aider à résorber ou à alléger une telle situation plus rapidement.

Ce chapitre présente et évalue TAPIOCA (*distribuTed and AdaPtive IntersectiOns Control Algorithm*). Cet algorithme utilise un réseau distribué (p. ex., un réseau de capteurs sans fil) afin d'obtenir les données de répartition des véhicules et de calculer et appliquer une politique de gestion des feux de circulation.

La section 4. 1 présente les algorithmes de gestion des feux s'appliquant sur les déploiements décrits au chapitre précédent. Nous présentons une architecture hiérarchique possible au niveau d'une intersection en section 4. 2. Nous détaillons ensuite, en section 4. 3, un premier algorithme de gestion des feux de circulation fonctionnant sur une intersection isolée. Cet algorithme sélectionne dynamiquement les mouvements et les temps de feux vert afin de réduire le temps moyen d'attente des automobilistes sans engendrer de famine. Dans la section 4. 4, nous présentons une généralisation de cet algorithme qui permet aux intersections proches de collaborer et de se synchroniser afin de créer des vagues vertes. L'algorithme général, TAPIOCA, est présenté dans la section 4. 5. Les résultats de simulation sont obtenus avec le

simulateur SUMO et sont présentés en section 4. 6. Ils comparent le temps moyen d'attente et les tailles des files d'attente générées par TAPIOCA, en comparaison aux algorithmes et méthodes similaires de l'état de l'art.

4. 1 Algorithmes de contrôle des feux de circulation

L'utilisation de la théorie des files d'attente est une approche venant en tête naturellement lorsqu'il s'agit de modéliser les files d'attente présentes aux intersections (section 2. 1. 2). Les auteurs de [ANR11] abordent par exemple son utilisation, qui permet d'obtenir – à partir des données recueillies des capteurs – la taille moyenne d'une file sur une voie. Le temps de feu vert est calculé par rapport à la taille de ces files, de manière à obtenir le temps nécessaire à les traiter.

En se basant sur un déploiement de deux capteurs par voie (section 3. 2), Yousef *et al.* [YAKS10] définissent un mécanisme de gestion des feux à une simple intersection en modélisant chaque mouvement comme une file d'attente M/M/1. En utilisant une matrice qui définit les mouvements en conflits, ils proposent un algorithme qui sélectionne les combinaisons de mouvements compatibles qui possèdent le plus grand nombre de véhicules sur leurs files. Ils calculent ensuite le temps de feu vert proportionnellement au nombre total de véhicules. Leur algorithme élabore ainsi un plan de feu dynamique cycle après cycle avec un nombre de phases qu'ils fixent à quatre au maximum. Ils étendent ensuite leur travail à un réseau maillé d'intersections en sélectionnant tout d'abord, lorsqu'une phase débute, les mouvements qui reçoivent le plus de véhicules en provenance d'intersections proches. Leur méthode prend en compte le temps nécessaire pour aller d'une intersection à une autre, en incluant les arrêts et les ralentissements.

Après avoir abordé un modèle distribué où les capteurs communiquent avec un contrôleur dans [TSS07], Tubaishat *et al.* proposent dans [TQSS08] une gestion des feux plus aboutie. Leur solution consiste à se baser sur un système de gain, qui correspond pour une voie au nombre de voitures entre les deux capteurs. Ainsi, chaque phase possible est évaluée par la somme des gains de ses mouvements. La phase qui est la plus chargée est donc sélectionnée en premier, jusqu'à ce que son nombre de véhicules passe en dessous d'une autre ou que le temps de feu maximum soit dépassé.

Zou *et al.* [ZYC09] se reposent sur un déploiement incorporant uniquement un capteur par direction. Ils supposent que ces capteurs sont capables de détecter les véhicules sur cinq mètres et utilisent des routes de deux voies dans leur scénario (soit 6,5 mètres de largeur). Ils sont ainsi capables de comptabiliser le nombre de voitures empruntant chacune des voies. Cela permet d'établir la durée des feux selon le nombre de véhicules par minute. Si le nombre de passages est inférieur, par exemple, à cinq véhicules par minute sur une direction, alors le feu correspondant est configuré à dix secondes. Ceci est une application de la logique floue. L'inconvénient majeur de cette solution est que le nombre de voies possibles pour une intersection est limité par la

portée de détection des capteurs. De plus, les mesures prises en compte sont basées minutes après minutes et non en fonction des phases ou des cycles.

Aucune de ces contributions n'autorise vraiment les mouvements entrant en conflit à avoir le feu vert en même temps. De plus, ils agissent généralement en créant des cycles. Il serait probablement plus intéressant d'agir phase après phase, afin de gagner en réactivité. Enfin, ils prennent uniquement en compte la taille des files d'attente, ce qui peut produire un problème d'ordonnancement bien connu : la famine. Dans ce dernier cas, le feu vert n'est jamais accordé à une file d'attente car elle n'est pas considérée prioritaire (c.-à-d., elle possède moins de véhicules que les autres). La figure 4.1, importée de l'un de nos premiers travaux [FCD12a], nous montre que jouer avec des critères de famine et des critères de file d'attente peut mener à des résultats plus intéressants qu'avec un seul objectif. L'algorithme présenté, dénoté "Actuated", est la première version de TAPIOCA. Nous pouvons voir qu'une sélection de phase basée à 75% sur la taille des files d'attente et à 25% sur un critère de famine procure le meilleur temps d'attente, en comparaison à d'autres configurations ou à l'algorithme de Yousef *et al.*. TMAX représente sur ce schéma le temps de feu vert maximum (dénoté $\tau^{g^{max}}$ dans cette thèse), qui dépend du scénario considéré mais qui – dans un cas général – peut être fixé aux environs de 30 secondes.

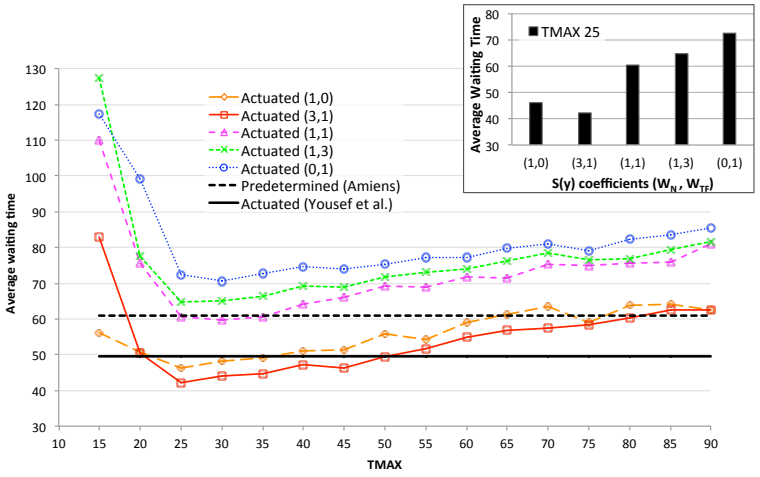


FIGURE 4.1 – Simulations extraites de [FCD12a] montrant l'évolution du temps d'attente en fonction de la pondération de deux critères : (file d'attente, famine). $S(y)$ représente une fonction de score équivalente à celle définie dans les sections suivantes.

Zhou *et al.* [ZCZW] proposent un algorithme qui sélectionne les combinaisons de mouvements qu'il est possible d'effectuer simultanément et sans conflits. Cela laisse le choix entre douze phases, comme l'illustre la figure 4.2 sur une intersection à quatre directions. Leur choix se base sur de multiples critères : la présence de véhicules prioritaires, la durée des périodes sans véhicule, le degré de famine, le temps d'attente total et la taille des files d'attente. Chaque critère est évalué et la sélection d'une phase se fait par priorité (p. ex., si il y a un véhicule d'urgence, donner le feu vert, sinon, passer au critère suivant). Toutefois, cet algorithme nécessite que tous les véhicules roulent à la même vitesse et soient du même type. [ZCW11] étend

ce travail au cas de plusieurs intersections en considérant davantage de paramètres. Un temps minimum de feu vert est calculé en se basant sur le nombre de véhicules qu'il y a à traiter localement. Il peut être étendu en fonction des flots qui arrivent des intersections voisines. Outre des hypothèses trop lourdes, le problème majeur de cet algorithme – à l'inverse des précédents – est sa complexité. En effet, les critères considérés nécessitent de la mémoire, des calculs et une fréquence de transmission élevée afin de maintenir des valeurs à jour.

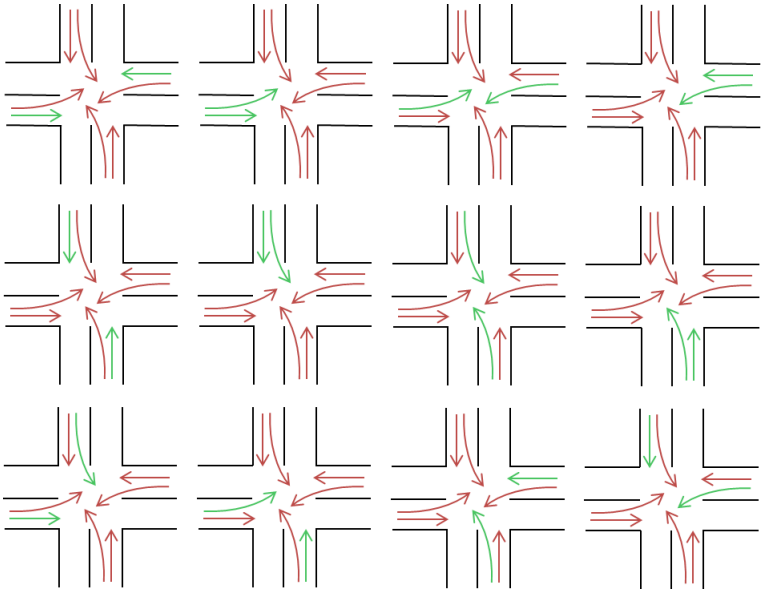


FIGURE 4.2 – Combinaisons sans conflits possibles pour un carrefour à 4 directions.

Dans tous ces papiers, les capteurs sont utilisés comme de simples détecteurs qui reportent leurs mesures à une entité centrale. Toutefois, une telle organisation ne passe pas à l'échelle et a une faible tolérance aux pannes. De plus, cette organisation possède une certaine latence dans la communication et la décision, car les données sont traitées et les décisions sont prises sur un point de collecte proche du contrôleur de feux.

4. 2 TAPIOCA : organisation du réseau

L'algorithme proposé dans ce chapitre, TAPIOCA, peut être exécuté sur toutes les architectures décrites au chapitre précédent (section 3. 2, figures 3.1 et 3.2). Par défaut, TAPIOCA distribue les tâches aux différents capteurs présents sur l'intersection.

Les nœuds de *destination* (DN) surveillent et comptabilisent les départs lorsque le feu est vert. Ces nœuds sont typiquement situés au niveau des feux de circulation, sur chaque voie d'entrée ou au début de chaque voie de sortie (cas de la figure 3.2(a)).

Les nœuds *source* (SN) mesurent les arrivées en permanence. Ils sont classiquement situés à une distance fixe des feux de circulation, sur chaque voie. Il est

important de bien régler cette distance. Si elle est trop petite, elle n'englobera pas suffisamment de véhicules. Si elle est trop grande, cela augmentera les erreurs liées, par exemple, aux changements de voies. Toutefois, les nœuds SN peuvent également être mobiles (p. ex., montés sur rail) ou peuvent être sélectionnés dynamiquement parmi un ensemble de nœuds fixes. Pour de grandes files d'attente, ils peuvent également être remplacés par des caméras.

4. 2. 1 Architecture

Parmi les nœuds DN, nous choisissons d'élire un *agrégateur de direction* pour chaque direction afin de collecter, traiter et agréger le trafic de tous les nœuds DN situés sur les voies émanant de la même direction. Ces agrégateurs servent également à la communication avec les intersections voisines, accessibles via leur route. Enfin, ils servent de niveau hiérarchique intermédiaire, afin d'éviter à une seule entité de tout récolter.

Ces agrégateurs communiquent avec un nœud responsable du calcul du plan de feux et de la communication avec le contrôleur de feux. Ce nœud *décision* peut être élu parmi les capteurs et réélu périodiquement pour remplir certains objectifs (p. ex., distribution des coûts énergétiques).

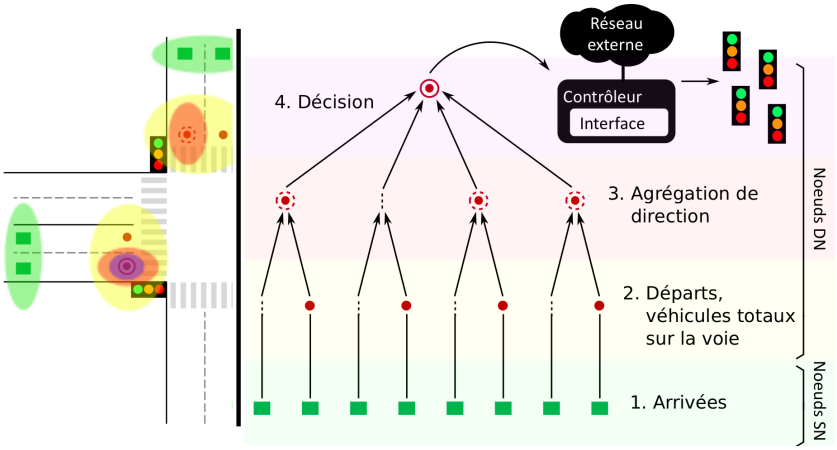


FIGURE 4.3 – Une intersection à quatre directions et son architecture.

Cette organisation, décrite sur la figure 4.3, n'est pas obligatoire pour TAPIOCA. Elle présente toutefois certains avantages.

Tout d'abord, les agrégateurs de direction possèdent des informations sur le nombre total de véhicules sortant de l'intersection vers une destination donnée. Ils sont en mesure d'envoyer cette information à toutes les intersections proches et sont de bons candidats afin d'agir en tant que passerelles vers ces intersections, dans le cas où le réseau fonctionne en multi-sauts.

Deuxièmement, tous les rôles d'agrégateurs sont échangeables. Cela signifie que le protocole d'élection distribué peut être en charge de sélectionner les meilleurs candidats en fonction de plusieurs critères (p. ex., niveau de batterie, capacité des nœuds, etc.). Dans la plupart des cas, ces critères sont numériques et il devient possible de

créer un score attaché à chaque nœud. Sur chaque nœud, la fonction génératrice de ce score serait monotone (toujours décroissante si nous prenons le cas du niveau de batterie). Cette propriété permet notamment d'appliquer des algorithmes d'élection ayant un coût raisonnable, tels que celui proposé par [JKZ02] qui opère en un temps de $\log(n)$.

Enfin, ces rôles peuvent être redéfinis entre les capteurs en cas de défaillances. Cette architecture procure un certain niveau de tolérance aux pannes.

4. 2. 2 Distance entre les nœuds SN et DN

Nous rejoignons bon nombre d'auteurs qui estiment que la distance séparant les nœuds SN et DN doit être fonction des limites temporelles fixées sur l'intersection [ZCZW]. Nous estimons ainsi qu'elle doit correspondre au nombre de véhicules qui peuvent traverser l'intersection lorsque la durée du feu vert est à son maximum autorisé, noté $\tau^{g^{max}}$. Cette durée est généralement bornée pour limiter le temps d'attente qui serait mal perçu par les usagers.

Cette distance peut être réglée à $n \cdot l_{veh}$, où n dénote le nombre de véhicules qui sont capables de passer en $\tau^{g^{max}}$ secondes et l_{veh} la taille moyenne d'un véhicule. Si τ^s est le temps de démarrage d'un véhicule et si τ^h représente le temps inter-véhiculaire qui sépare deux véhicules, alors :

$$n = \frac{\tau^{g^{max}} - \tau^s}{\tau^h} \tag{4.1}$$

Pour rappel, selon [Gor+05], $l_{veh} = 6$ mètres, $\tau^s = 4\,s$ et $\tau^h = 2\,s$ sont des valeurs réalistes qui peuvent être ajustées empiriquement. Pour plus de clarté, ces notations sont illustrées en annexe A.

En se basant sur des études précédentes [FCD12a ; FCD12b] et sur [Gor+05], une valeur typique pour cette distance serait 75 mètres. Selon l'équation 4.1, cela correspond à une valeur de $\tau^{g^{max}}$ de 30 secondes, ce qui est cohérent par rapport à nos simulations (p. ex., figure 4.1).

4. 3 TAPIOCA : le cas d'une intersection isolée

Dans [FCD12a], nous considérons que chaque intersection est seule, isolée. Dans ce cas de figure, il n'y a pas besoin de communication à large échelle, le but étant juste de considérer un flux de véhicules entrant. Les capteurs déployés sur chaque intersection échangent des informations en utilisant des communications sans fil et se mettent en accord sur un plan de feux.

Au lieu de définir des cycles, l'algorithme fonctionne uniquement avec une granularité d'une phase. De plus, la sélection dynamique des mouvements s'effectue sur la base de deux objectifs : réduire le temps d'attente moyen et éviter les situations de famine. Enfin, cet algorithme permet également de donner le feu vert aux mouvements qui sont légèrement en conflit. Ces mouvements sont des mouvements qui

n'ont pas d'impact sur la sécurité des utilisateurs, c.-à-d. lorsque des règles de priorités évidentes existent entre les mouvements. Ces trois propriétés permettent à cet algorithme de réduire le temps moyen d'attente des usagers.

L'algorithme 4.1 décrit en pseudo-code la décision qui est prise par le nœud responsable du calcul du plan de feux, afin de gérer une intersection isolée. Il est détaillé et expliqué au fil de cette section. Les mouvements sont d'abord évalués à l'aide de plusieurs objectifs (lignes 1 à 8, section 4. 3. 1). Ensuite, une phase est sélectionnée ou créée en associant les mouvements existant, en tenant compte d'éventuels conflits avec d'autres voies ou avec des piétons (lignes 9 à 17, sections 4. 3. 2, 4. 3. 3 et 4. 3. 4). Enfin, la durée des feux verts est calculée en fonction de l'importance du trafic et la phase est lancée (lignes 18 à 21, section 4. 3. 5).

4. 3. 1 Classement de mouvements sur la base de deux objectifs

Considérons un mouvement possible (s, d) allant de la direction s à la direction $d \in D$, D dénotant l'ensemble des directions possibles. Le nœud de décision (c.-à-d., couche 4 sur la figure 4.3) connait la répartition des véhicules sur l'intersection. Il est donc capable d'associer un score local $S(s, d)$ au mouvement (s, d). Ce score dépend du nombre de véhicules présents sur les voies entrantes qui composent le mouvement $(\eta^{(s,d)})$ et du temps passé depuis la dernière sélection du mouvement $(F^{(s,d)})$, c.-à-d. depuis qu'il a eu le feu vert.

Nous devons combiner les métriques qui reflètent chaque objectif (charge et intervalle entre deux sélections successives) dans une seule expression afin de définir formellement ce score. À ces fins, nous normalisons chaque objectif en utilisant une fonction générique, $\gamma()$. Le but de cette fonction est de mettre les objectifs F et η dans l'intervalle $[0; 1]$ et ainsi de les dépourvoir de dimensions, à des fins de comparaisons. $\gamma(\{F, \eta\}^{(s,d)})$ est défini naturellement comme le ratio entre la valeur de l'objectif du mouvement (s, d) et la somme de toutes les valeurs d'objectif des différents mouvements :

$$\gamma(F^{(s,d)}) = \frac{F^{(s,d)}}{\sum F} \qquad \gamma(\eta^{(s,d)}) = \frac{\eta^{(s,d)}}{\sum \eta}.$$

Notons que si la somme toutes les valeurs de $\{F, \eta\}$ est nulle, alors ce score est nul également. Lorsque cette étape est réalisée, une approche classique serait de définir le score comme étant une combinaison linéaire de $\gamma(F^{(s,d)})$ et de $\gamma(\eta^{(s,d)})$. Toutefois, sachant que nous cherchons un classement plutôt qu'une évaluation, nous aimerions donner plus de poids aux mouvements qui ont un nombre de véhicules nettement plus élevés que les autres, ou qui n'ont pas été sélectionnés depuis longtemps. C'est pourquoi nous définissons le score comme étant une somme pondérée des carrés des mesures normalisées :

$$S(s, d) = W_\eta \cdot \left(\gamma(\eta^{(s,d)})\right)^2 + W_F \cdot \left(\gamma(F^{(s,d)})\right)^2,$$

Algorithme 4.1: algorithme simplifié de TAPIOCA sur une intersection isolée

Paramètres : $mouvements$, liste des mouvements de l'intersection ; $phases$, liste des phases possibles sur l'intersection ; $\tau^{g^{max}}$, temps de feu vert maximal.

Variables : η^m, nombre de véhicules sur le mouvement m ; η^{voie}, nombre de véhicules sur la voie $voie$; F^m, dernière fois que le mouvement m a eu le feu vert ; $p_{suivant}$, phase suivante.

```
// Évaluation de chaque mouvement
majF();
pour chaque m ∈ mouvements faire
    η^m ← 0;
    pour chaque voie ∈ m faire
        η^m ← η^m + (η^voie / nbMouvements(voie));
    finprch
    S(m) = (η^m / Ση)^2 + (F^m / ΣF)^2;
finprch
// Sélection d'une phase parmi celles connues (p. ex., SUMO - Sec. 4. 3. 2. 2)
S_max = −1 ; p_suivant = −1 ; η^max = −1;
pour chaque p ∈ phases faire
    // Le score d'une phase est la somme du score de ses mouvements
    score = Σ_{m∈p} S(m);
    // Une phase est sélectionnée si elle a le plus haut score et si elle
       possède des véhicules
    si S_max < score et fileMax(p) > 0 alors
        S_max ← score;
        p_suivant = p;
        η^max = fileMax(p);
    finsi
finprch
// Calcul de la durée du feu vert et lancement de la phase
si S_max < score et fileMax(p) > 0 alors
    τ^g = min(η^max · τ^h, τ^{g^max});
    lancerPhase(p_suivant, τ^g);
finsi
```

---

majF() : Remet à jour les valeurs de F^m, pour chaque mouvement m. Au moment de l'appel de cette procédure, tous les mouvements ayant le feu vert sont paramètres avec une valeur nulle ($F^m = 0$). Tous les autres sont mis à jour en fonction du dernier temps depuis lequel ils ont eu le feu au vert ($F^m = temps\ actuel - dernier\ temps\ connu$).

nbMouvements(voie) : retourne le nombre de mouvements possibles depuis une voie $voie$.

fileMax(phase) : retourne la taille de la plus grande file appartenant à la phase $phase$.

lancerPhase(phase, temps) : Provoque l'arrêt de la phase en cours et lance la phase suivante $phase$ pendant une durée de $temps$ secondes.

où W_η et W_F sont des poids définis par le concepteur, qui peuvent être changés empiriquement par les opérateurs afin de favoriser la performance (temps moyen d'attente) ou l'expérience de l'utilisateur (famine). Dans l'article [FCD12a] (dont l'un des résultats est représenté sur la figure 4.1), nous montrons par simulation l'intérêt de confronter ces deux paramètres. Il est également possible de mettre en place un algorithme d'apprentissage local ou un réglage dépendant des poids des intersections voisines. Par défaut, ces poids restent identiques. Mettre au carré les objectifs garde les valeurs dans le même intervalle mais accentue leur importance quand celles-ci sont élevées. Cette dernière propriété a l'effet désiré : si l'un des objectifs a une valeur élevée pour un mouvement donné par rapport aux autres, sa contribution au score doit être suffisamment importante pour favoriser sa sélection. Si aucun véhicule n'est présent sur les voies originaires de la direction s, le score est défini comme étant nul pour tous les mouvements concernés, même si le mouvement n'a pas été sélectionné depuis longtemps : $\forall d \in D, S(s,d) = 0$.

4. 3. 2 Sélection d'une phase

4. 3. 2. 1 Basée sur une matrice des conflits

Une fois que les scores ont été calculés pour chaque mouvement, le nœud de décision examine quels mouvements peuvent être combinés afin de créer une nouvelle phase. À ces fins, il utilise une matrice des conflits, qui indique les mouvements qui peuvent être effectués en toute sécurité simultanément (sec. 2. 2. 2. 1).

Cette matrice est utilisée pour combiner les mouvements individuels qui peuvent composer une phase valide, en additionnant leurs scores et en explorant toutes les combinaisons possibles (afin de trouver les plus optimales). La phase sélectionnée est celle qui possède le plus haut score.

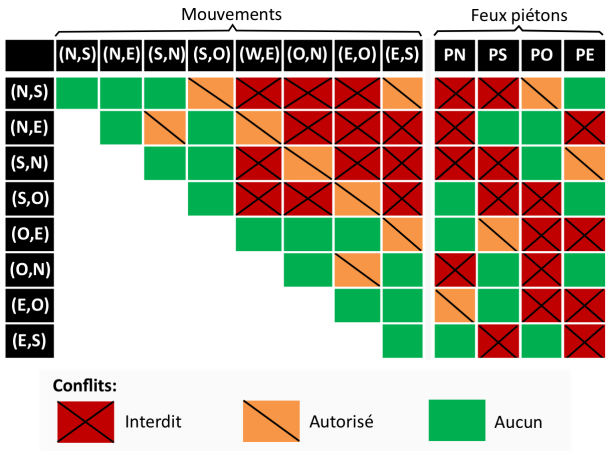


FIGURE 4.4 – Exemple d'une matrice des conflits.

La figure 4.4 représente une telle matrice des conflits avec trois niveaux. (1) Les éléments verts indiquent les mouvements non conflictuels. (2) Les éléments rouges indiquent les mouvements qui ne peuvent pas se dérouler en même temps, p. ex. pour des raisons de sécurité. (3) Entre les deux, les éléments oranges indiquent les mouve-

ments qui sont en conflit, mais avec un niveau de sécurité acceptable (p. ex. priorité face aux mouvements allant à gauche). Ce cas est représenté sur la figure 4.5, où deux mouvements de virage à gauche sont en conflit. Notons que ce cas peut cependant limiter le nombre de véhicules qui peuvent traverser l'intersection et potentiellement introduire des situations de blocage. La section 4. 3. 3 décrit comment TAPIOCA permet de gérer de telles situations.

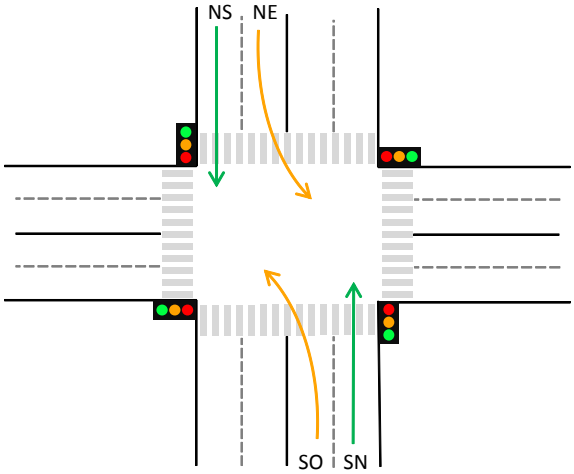


FIGURE 4.5 – Exemple de phase avec deux mouvements en conflit léger.

Pour finir, remarquons que cette matrice comprend également des colonnes pour les passages piétons, abordés en section 4. 3. 4.

4. 3. 2. 2 Basée sur une phase statique

Trouver la meilleure combinaison de mouvements lorsque la matrice des conflits autorise de multiples mouvements simultanés requiert des calculs. Il nous faut en effet maintenir une liste de toutes les combinaisons de mouvements possibles sur chaque intersection. Une combinaison pouvant avoir plus de deux mouvements. Cela nécessite des calculs initiaux de complexité variable (dépendant de la taille de la matrice des conflits) et des coûts de mémoire supplémentaires. Pour plus de simplicité, dans nos simulations (voir 4. 3. 6, 4. 6. 1 et 4. 6), nous supposons que la liste des phases possibles est déjà connue. Pour cela, nous nous basons sur les plans de feux statiques préconfigurés sur chaque intersection, dont les phases sont réalistes. Afin de composer des phases en conflit léger (section précédente), nous pouvons également combiner les phases des plans de feux statiques.

Pour chaque phase possible, il nous est facile de calculer le score $S(s, d)$ de chaque mouvement ayant le feu vert. La phase sélectionnée est la phase qui possède la plus haute somme de scores.

4. 3. 3 Politique avec des mouvements faiblement en conflit

Les véhicules qui tournent à gauche ont généralement la priorité la plus basse, ils sont bloqués lorsqu'un véhicule vient du mouvement situé dans la direction opposée. La possibilité de laisser les deux mouvements se produire en même temps dépend donc de la charge des deux mouvements.

Les stratégies classiques consistent à laisser passer chacun de ces mouvements dans une phase dédiée, ou de séparer la phase en deux sous-phases : la première sous-phase permet aux mouvements concurrents de passer et la seconde permet aux mouvements à faible priorité de passer. Une telle phase est légèrement différente du cas où deux phases s'enchainent, car la deuxième sous-phase s'enchaine immédiatement après la première. TAPIOCA adopte cette deuxième solution.

4. 3. 4 Passages piétons

Les passages piétons peuvent facilement être inclus dans le modèle de TAPIOCA, en ajoutant des entrées à la matrice des conflits, comme le montre la figure 4.4. Les passages pour piétons peuvent être considérés comme des voies de véhicules classiques. Cependant, ils se comportent différemment, car il est plus difficile de dénombrer les usagers en attente. TAPIOCA adopte une autre approche : lorsque la composition d'une phase est déterminée, tous les passages piétons qui ne sont pas en conflit avec les mouvements sélectionnés sont sélectionnés également. Fonctionner ainsi ne génère pas de famine car les feux piétons suivent le rythme des feux de circulation. Cette règle peut être allégée, nous pourrions par exemple tolérer les conflits modérés avec une règle de priorité (p. ex., lorsque des véhicules tournent à droite et doivent donner la priorité aux piétons). La matrice des conflits doit être configuré pour permettre à tous les passages pour piétons d'avoir le feu vert. Dans ce cas, les passages pour piétons ne sont pas un obstacle à la composition de phase, ils sont complémentaires et n'interfèrent pas dans le processus de décision.

Dans des cas particuliers, il est impossible de configurer une matrice des conflits avec aucun conflit entre les véhicules et les piétons. Dans ce cas, un passage pour piétons P_n peut avoir les mêmes propriétés que les mouvements de véhicules : F^{P_n} est la dernière fois que le feu pour piétons est passé au vert et η^{P_n} est un indicateur de présence de piétons. Ce dernier peut être calculé avec plus ou moins de précision par des caméras, des capteurs piézoélectriques ou plus simplement avec des boutons poussoirs.

4. 3. 5 Durée de phase

La vie d'une intersection est, dans TAPIOCA, une succession de phases. Contrairement à la plupart des stratégies, il n'y a pas de notion explicite de cycle. Les phases se succèdent et TAPIOCA veille à ce que tous les mouvements soient sélectionnés régulièrement.

Quand une phase se termine, les feux passent à la couleur orange pendant un temps τ^{orange}, traditionnellement fixé à 3 secondes. Bien entendu, des méthodes précises d'obtention de ce temps existent, nous le simplifions toutefois à une valeur moyenne pour éviter des temps de calcul trop longs [Gor+05 ; US 08 ; Min09]. Puis, tous les feux se mettent au rouge pendant un temps de sécurité τ^{secure}. Cela laisse suffisamment de temps au contrôleur afin de déterminer quels mouvements composeront la phase à venir.

Une fois les mouvements sélectionnés, en adéquation avec les critères exposés dans les sections précédentes, le contrôleur calcule le temps de feu vert de la phase.

4. 3. 5. 1 Temps de feu vert

Pour rappel, T_s est le temps de démarrage et τ^h est le temps qui sépare le passage de deux véhicules successifs. $\tau^{g^{max}}$ est la durée de feu vert maximale, qui est soit défini par l'utilisateur, soit déterminé de façon à équilibrer les performances et l'expérience des utilisateurs (section 4. 2). τ^g, le temps de feu vert, est calculé en fonction de la charge de la voie ayant le plus grand nombre de véhicules. Ceci laisse la possibilité à l'intersection de pouvoir vider toutes les voies concernées :

$$\tau^g = min(\tau^s + \eta^{max} \cdot \tau^h,\ \tau^{g^{max}}),$$

où η^{max} est le nombre de véhicules sur la voie la plus chargée. Ce temps est borné par $\tau^{g^{max}}$ afin d'éviter de bloquer l'intersection. Dans le cas où $\tau^g < \tau^{g^{max}}$, des véhicules additionnels peuvent arriver sur ces voies. Dans ce cas, nous choisissons de laisser le temps de feu vert se rallonger d'une période de τ^h, jusqu'à ce qu'il n'y ait plus de nouvelles arrivées, ou jusqu'à ce que $\tau^g = \tau^{g^{max}}$.

4. 3. 5. 2 Effets du temps de feu maximum

La valeur de $\tau^{g^{max}}$ a une influence sur la performance globale du système. Dans [FCD12a] (dont l'un des résultats est présenté sur la figure 4.1), nous trouvons que le temps optimal de $\tau^{g^{max}}$ pour une intersection de la ville d'Amiens (France) est situé entre 25 s et 35 s. La valeur de $\tau^{g^{max}}$ est plus petite lorsque les conflits sont interdits, car laisser tous les mouvements survenir dans cette situation nécessite plusieurs phases, ce qui réserve moins de temps à une seule phase. Dans ces premières simulations, nous avons choisi de paramétrer empiriquement $\tau^{g^{max}}$, en testant un ensemble de valeurs de 15 s à 70 s et en utilisant le meilleur résultat.

Si une telle approche empirique est toujours possible, nous proposons ici de régler $\tau^{g^{max}}$ en considérant la durée initiale du cycle (τ^c) défini dans les plans de feux statiques lorsqu'ils sont disponibles.

τ^c est notamment réglé pour limiter le temps d'attente entre deux sélections de la même phase, et est défini en fonction du trafic et de la configuration de l'intersection. Sur une intersection arbitraire, nous configurons $\tau^{g^{max}}$ proportionnellement au nombre de véhicules qui composent la phase :

$$\tau^{g^{max}} = \frac{\eta^{max} \cdot \tau^{c}}{\sum_{\{a,b\} \in D} \eta_{max}^{(a,b)}}.$$

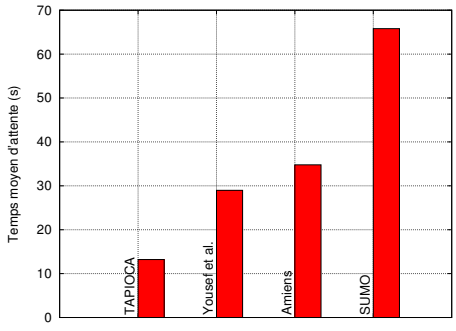
Ainsi, nous sommes assurés d'obtenir une valeur de $\tau^{g^{max}}$ cohérente par rapport à la diversité des intersections. Cette méthode est appliquée dans nos simulations, et se trouve utile et efficace pour les grands scénarios tels que TAPASCologne (section 4. 6).

4. 3. 6 Évaluation sur une intersection isolée (Amiens)

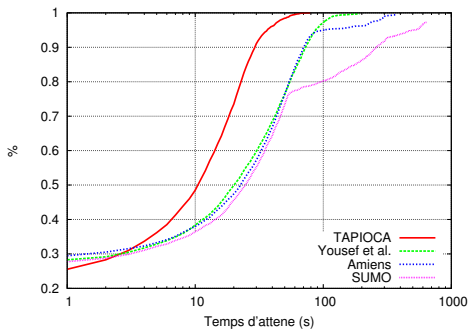
Les simulations présentées sur la figure 4.6 sont effectuées à l'aide de SUMO et sont basées sur une intersection de la ville d'Amiens (France), pour laquelle nous avons à notre disposition des données réelles de trafic, la répartition des véhicules et les plans de feux statiques (cycles, phases et temporisations). Chaque simulation dure 7 200 secondes et ne nécessite aucune période d'entrainement. Les résultats présentés sur la figure 4.6 sont les valeurs moyennes calculées sur les véhicules qui traversent l'intersection pendant ce temps de simulation.

Les figures 4.6(a) et 4.6(b) montrent que TAPIOCA réalise le meilleur temps d'attente moyen par rapport à un algorithme adaptatif issu de la littérature (55%), au plan de feu statique de la ville d'Amiens (62%) et au plan de feux généré par SUMO (80%). Notons que SUMO génère ses plans de feux sans prendre en compte le trafic qui est potentiellement présent, mais en se basant uniquement sur la cartographie du réseau et des valeurs représentatives des cas de figures généraux. Les plans de feux générés sont donc souvent inadaptés et représentatifs d'un scénario mal configuré. La figure 4.6(c) montre que TAPIOCA produit également les files d'attente les plus courtes en moyenne. Enfin, la figure 4.6(d), qui décrit la répartition de la meilleure stratégie pour les véhicules intégrés aux simulations, nous montre que TAPIOCA génère un meilleur temps d'attente pour plus de 40% des véhicules du scénario.

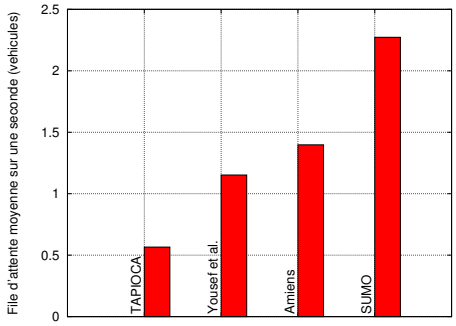
Les résultats que nous décrivons sur une intersection semblent encourageants. Toutefois, nous souhaitons nous intéresser à présent aux effets d'un scénario possédant plusieurs intersections. Notre algorithme ayant uniquement une vision locale, aucune collaboration n'existe entre les feux verts d'intersections adjacentes. Ceci peut notamment provoquer des arrêts ou des phases qui enverraient de manière excessive, par exemple, des véhicules sur une intersection voisine encombrée. Cette intuition est notamment levée par la figure 4.7, qui est extraite de simulations présentant un scénario à plusieurs intersections, abordé dans la suite du document. Nous constatons que la version isolée de TAPIOCA se comporte nettement moins bien que l'algorithme de Yousef *et al.*. (conçu pour gérer plusieurs intersections) et à hauteur du plan de feux prédéterminé de la ville d'Amiens (conçu pour faire correspondre les feux de circulation).



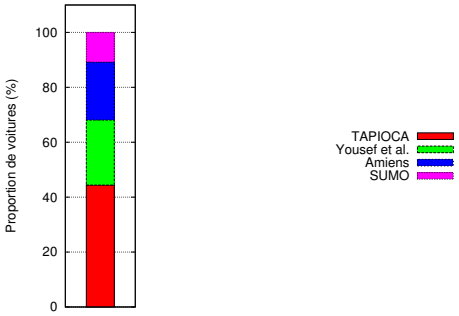
(a) Temps moyen d'attente.



(b) CDF du temps d'attente (échelle logarithmique.



(c) Taille moyenne des files de chaque voie.



(d) Répartition de la meilleure stratégie.

FIGURE 4.6 – Simulations sur une intersection isolée de la ville d'Amiens.

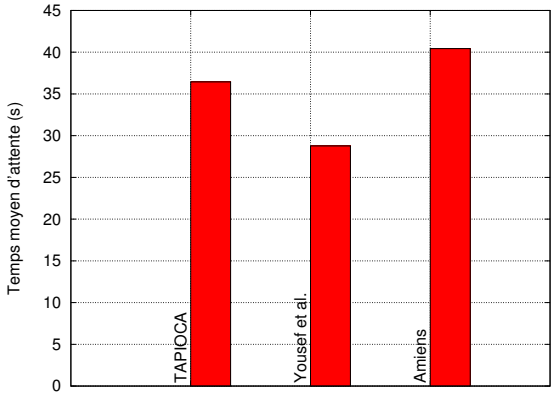


FIGURE 4.7 – Simulations sur trois intersections adjacentes de la ville d'Amiens.

4. 4 TAPIOCA : le cas de plusieurs intersections

Dans cette section, nous proposons une généralisation de l'algorithme précédent à plusieurs intersections adjacentes, en définissant une méthode pour synchroniser les intersections proches et créer des vagues vertes. Nous supposons que chaque intersection peut communiquer avec ses intersections adjacentes à travers le réseau de capteurs sans fil. Des relais supplémentaires peuvent être utilisés, formant un réseau multi-sauts, lorsque la distance entre deux intersections est plus grande que la distance de transmission des nœuds, comme étudié au chapitre 3. Une connexion WAN est également possible, si elle est disponible. Nous supposons également que les horloges des nœuds sont synchronisées avec une précision suffisante pour l'application (c.-à-d. approximativement une seconde).

4. 4. 1 Cas 1 : amélioration du score

Dans [FCD12b], nous avons proposé une première approche pour gérer le cas des intersections multiples. Cette approche sélectionne les mouvements qui composent la phase à venir sur la base de mesures locales et distantes combinées en un score global, qui mélange trois objectifs. D'abord, un score local est calculé avec la méthode décrite dans la section 4. 3. 1. Ensuite, les résultats sont échangés entre intersections proches de sorte à favoriser les mouvements qui, (1) ont la plus grande capacité (c.-à-d. éviter d'envoyer trop de véhicules dans les zones déjà embouteillées) et qui, (2) tentent de synchroniser les intersections successives pour créer des vagues vertes.

Les vagues vertes sont des chemins de feux verts successifs, synchronisées afin que les véhicules ne ralentissent pas. Il s'agit de l'une des techniques les plus efficaces pour décharger un réseau, car elle rend le trafic plus fluide. La mise en œuvre habituelle de vagues vertes suppose que le véhicule roule à la vitesse limite et la synchronisation doit être adaptée lorsqu'une surcharge apparait (sec. 2. 2. 3. 5). Cette première version ne tient pas compte de la charge et sa performance n'est donc pas optimale lorsque le niveau de congestion retarde considérablement les véhicules entre deux intersections voisines.

4. 4. 2 Cas 2 : synchronisation indépendante

Dans la version de TAPIOCA introduite dans le présent chapitre, nous avons choisi de modifier la philosophie de synchronisation des intersections. Au lieu de prendre directement en compte les informations provenant de voisins dans le calcul d'un score complexe, nous laissons chaque intersection prendre ses propres décisions au niveau local, comme si elle était isolée. Le résultat est un ensemble de mouvements actifs et un nombre prévu de véhicules qui iront, pendant la phase suivante, de l'intersection de départ (I^1) jusqu'à certaines intersections voisines.

Notons I^2 l'une des ces voisines qui devrait s'attendre à recevoir un flux entrant. I^1 pourrait envoyer un message afin d'avertir I^2 à propos de ce nouveau flux. Cependant, cela pourrait générer un trafic réseau élevé, qu'il nous faut limiter afin de

réduire la probabilité de pertes et de congestion dans le réseau de communication. De plus, l'utilisation d'un réseau de type IEEE 802.11p, qui a vocation à être partagé, accentuerait ce problème. C'est pourquoi nous avons choisi de limiter la transmission de ces messages dans le seul cas où I^2 est moins chargée (c'est à dire a un nombre total de véhicules plus faible) que I^1. L'idée derrière ce filtrage est une stratégie gloutonne : I^2 ne devrait envisager l'interruption de son propre cycle que si cela permet de réduire la charge d'une intersection voisine plus chargée. Sinon, elle poursuit son propre programme local.

Ce message de synchronisation contient le nombre de véhicules que I^1 s'attend à envoyer à I^2 et une estimation du temps requis par ces véhicules pour atteindre I^2. Ce temps peut être évalué en se basant simplement sur la distance entre les deux intersections. Il peut également être évalué par l'échange d'un échantillon de signatures électromagnétiques des véhicules entre les nœuds DN de I^1 et les nœuds SN de I^2, en tenant compte du temps inter-véhiculaire (τ^h), orange (τ^{orange}), de sécurité (T^{secure}) et de démarrage (T^s) (sec. 2. 2. 3).

Lorsque I^2 reçoit un tel message, il évalue localement la possibilité de rompre le cycle en cours pour favoriser une vague verte. En effet, le filtrage de message qui a été effectué par I^1 repose nécessairement sur des informations légèrement anciennes. Il compare les deux gains attendus, c'est à dire le nombre de véhicules qui traversent l'intersection si la phase actuelle est maintenue et si la vague verte interrompt la phase actuelle. Si le choix est en faveur de la vague verte, I^2 détermine une nouvelle composition de la phase, en appliquant l'algorithme de TAPIOCA classique avec la contrainte supplémentaire de sélectionner nécessairement les mouvements qui correspondent au flux entrant.

Les messages de synchronisation arrivent de manière asynchrone sur une intersection, qui pourrait avoir à faire face à plusieurs demandes au cours de la même phase. Dans ce cas, la stratégie qui maximise le nombre de véhicules à recevoir est sélectionnée.

La figure 4.8 représente un réseau de neuf intersections. Chacune de ces intersections déroule une phase, composée des mouvements représentés par les flèches vertes. Les messages de synchronisation qui sont transmis sont représentés par les flèches noires. Remarquons qu'une intersection n'envoie pas de message à des voisins qui ne sont pas la destination des mouvements de la phase courante, ni à des voisins qui sont plus chargés en termes de véhicules. Une fois que ces messages sont reçus, les intersections de destination prennent une décision et ignorent les demandes lorsque le gain des objectifs locaux est plus intéressant que le gain des vagues vertes. Ces demandes ignorées sont représentées par des flèches barrées.

4. 5 TAPIOCA : algorithme détaillé

TAPIOCA est distribué sur les intersections et sélectionne les mouvements qui composent la phase $P + 1$ à la fin de la phase P, en se basant sur les principes exposés aux sections 4. 3 et 4. 4. L'algorithme détaillé ci-dessous est destiné à être mise en place sur l'architecture 3.1 ou 3.2(b), c.-à-d. avec les nœuds SN positionnés

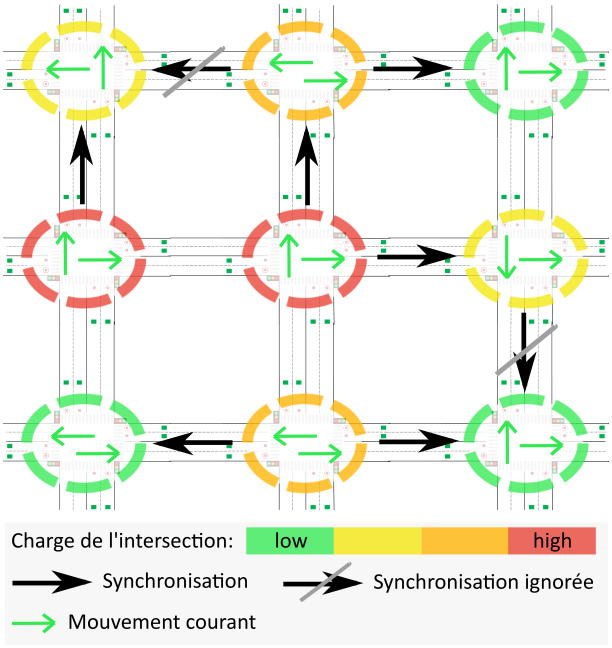


FIGURE 4.8 – Exemple de scénario où des messages de synchronisation envoyés entre les intersections.

sur les mêmes voies que les nœuds DN. Le cas de l'architecture 3.2(a) est détaillé dans [FCD12b].

TAPIOCA est décomposé en 7 étapes :

1. **Comptabilisation des véhicules sur chaque voie (nœuds SN et DN) :** pour chaque voie i, à la fin de la phase P, chaque nœud SN (couche 1) envoi $\eta_i^A(P)$, le nombre d'arrivées enregistrées pendant la phase P, au nœud DN qui est sur sa voie (couche 2). En parallèle, chaque nœud DN a enregistré le nombre de départs survenus durant la phase ($\eta_i^D(P)$). Chaque nœud DN met ensuite à jour $\eta_i(P+1)$, le nombre de véhicules qui sont présents sur la voie au début de la phase $P+1$:

$$\eta_i(P+1) = \eta_i(P) + \eta_i^A(P) - \eta_i^D(P)\,.$$

   Il transmet ensuite toutes les valeurs de $\eta_i(P+1)$ à l'agrégateur de direction (couche 3) qui gère la direction entrante à laquelle le nœud DN appartient.

2. **Agrégation par direction (nœuds agrégateurs de direction) :** pour chaque direction s, l'agrégateur maintient le temps passé depuis la dernière sélection des mouvements qui commencent en s, F^s, afin de détecter et de prévenir les cas de famine. Pour chaque mouvement commençant en s, il additionne les valeurs de η_i^{P+1} reçues depuis toutes les voies pertinentes pour obtenir $\eta^{(s,d)}$, la longueur totale des files d'attentes pour le mouvement (s,d).

   Si une voie i est l'origine de M mouvements ($M > 1$), nous choisissons d'établir par défaut pour chaque mouvement (s,d), $\eta^{(s,d)} = \eta_i^{P+1}/M$, afin d'éviter de

compter M fois les véhicules de la voie i. Si des capteurs additionnels sont installés sur les voies de sortie, ils peuvent coopérer avec le nœud DN de la voie qui a plusieurs mouvements afin de déterminer un coefficient pour chaque mouvement. Par exemple, si une moyenne de 60% de véhicules de la voie i suivent le mouvement (s, d), alors nous pouvons configurer $\eta^{(s,d)} = \eta_i^{P+1} \cdot 0.6$.

Finalement, le nœud responsable de la direction transmet ces deux valeurs au nœud responsable de l'intersection (couche 4). Ces types de nœuds gèrent également les messages de synchronisation reçus des intersections voisines. Si ils en ont, ils les transmettent au même moment.

3. **Composition de la prochaine phase (nœud décision) :** une fois qu'il a reçu les données de tous les agrégateurs, le nœud de décision calcule les scores locaux $S(s, d)$ des différents mouvements (section 4. 3. 1). Il combine ensuite les mouvements en utilisant la matrice des conflits ou un ensemble prédéterminé de phases du plan de feux statique et sélectionne la combinaison qui possède le meilleur score. À ce stade, des critères supplémentaires peuvent être envisagés (p. ex., la détection de véhicules d'urgence, éviter une combinaison en cas de détection d'accident). Un exemple de cette opération, en pseudo-code, est donné sur l'algorithme 4.1 (lignes 1 à 17).

4. **Durée de la prochaine phase (nœud décision) :** une fois que les mouvements sont sélectionnés, le temps de feu vert est configuré en accord avec la section 4. 3. 5.

5. **Transmission aux intersections voisines (nœud décision, agrégateurs) :** le nœud décision transmet la composition de la prochaine phase et sa temporisation aux nœuds agrégateurs, avec un éventuel message de synchronisation et le nombre de véhicules sur l'intersection. Chaque nœud agrégateur transmet ensuite ces informations aux intersections voisines.

6. **Application de la phase (nœud décision) :** le nœud décision ordonne au contrôleur de changer les feux spécifiés pendant le temps spécifié. Cela marque le début de la phase $P + 1$.

7. **Pendant la phase $P+1$ (nœud décision).** Si un message de synchronisation arrive et est considéré intéressant (section 4. 4), le nœud décision relance le processus de sélection de phase, en forçant la sélection des mouvements évoqués par la vague verte. Lorsque la phase est finie, le nœud décision applique si nécessaire le temps supplémentaire pour les voies tournant à gauche (section 4. 3. 3).

- **Surveillance des véhicules entre les intersections (nœud DN) :** durant la phase $P + 1$, les nœuds DN des directions sélectionnées peuvent envoyer les signatures temporisées des véhicules aux nœuds DN correspondant sur les intersections qui sont susceptibles de les accueillir. Cela permet d'estimer le temps qui est requis pour aller d'une intersection à une autre. Ce délai peut être utilisé afin d'améliorer le processus de création de vagues vertes, sa variation permettant de détecter assez tôt une formation d'embouteillage.

La figure 4.9 représente le diagramme de collaboration de TAPIOCA, c.-à-d.. les relations qui existent entre les nœuds et les principales actions de l'algorithme. Notons que l'algorithme se base sur des notions définies dans ce chapitre. Toutefois, il est facile d'imaginer des améliorations. Nous pourrions par exemple considérer la défaillance d'un capteur ou des informations provenant d'un réseau véhiculaire, de téléphones portables ou de GPS connectés.

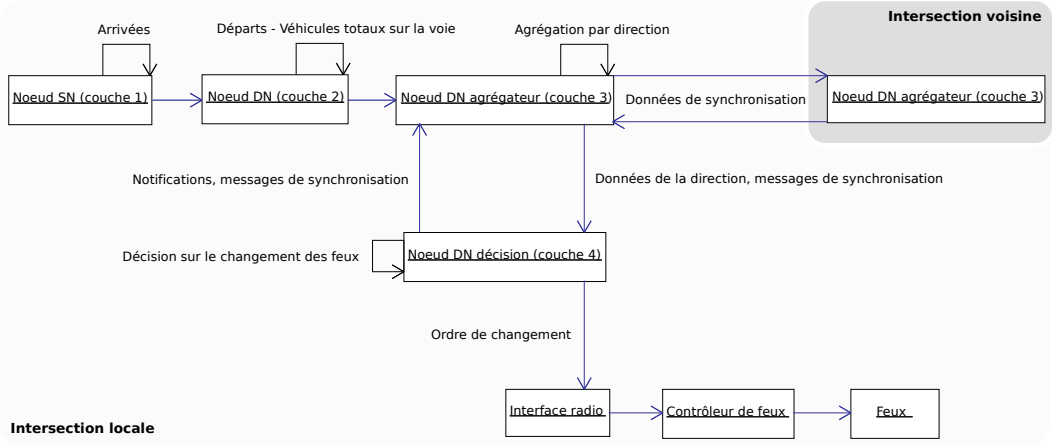


FIGURE 4.9 – Diagramme de collaboration de TAPIOCA.

4. 6 Simulations

Nous évaluons TAPIOCA en utilisant le simulateur SUMO 0.17. Nos simulations sont faites sur plus de *1 425 137* voyages distribués parmi plus de *32* scénarios distincts : les résultats complets et plus de figures, sont disponibles en ligne [1]. Nous détaillons dans cette section des métriques et scénarios représentatifs.

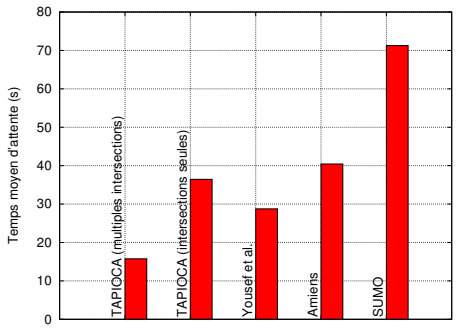
4. 6. 1 Scénario 1 : ville d'Amiens

Les simulations présentées sur les figures 4.10, de la même manière que celles présentées en section 4. 3. 6, sont basées sur trois intersections de la ville d'Amiens (France), pour laquelle nous avons des données de trafic, dont la répartition des véhicules et les plans de feux (cycles, phases, synchronisations entre intersections et temps de feux). Chaque simulation dure 7 200 s et est traversée par plus de 5 500 véhicules.

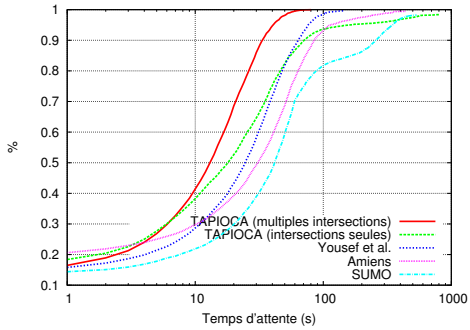
Les figures 4.10(a) et 4.10(b) montrent la comparaison entre TAPIOCA, TAPIOCA sur une intersection isolée ([FCD12a], sec. 4. 3), l'algorithme de Yousef *et al.* ([YAKS10]), le plan de feux utilisé sur Amiens et le plan de feux généré par SUMO.

Les résultats de simulations nous montrent que TAPIOCA, en créant des vagues vertes, accompli le meilleur temps moyen d'attente. Il est de 56% meilleur que TA-

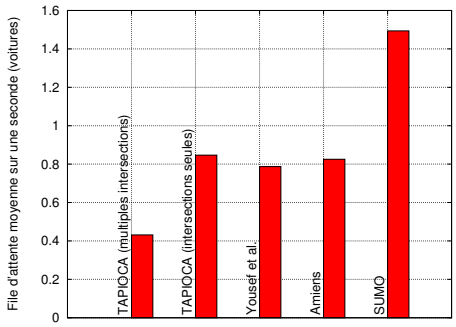
1. http://tapioca.sfaye.com/



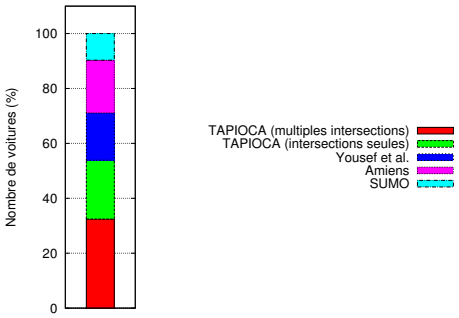
(a) Temps moyen d'attente.



(b) CDF du temps d'attente (échelle logarithmique).



(c) Longueur de file moyenne.



(d) Répartition de la meilleure stratégie.

FIGURE 4.10 – Comparaison de performances : plusieurs intersections de la ville d'Amiens.

PIOCA sur simple intersection, 45% meilleur que l'algorithme de Yousef *et al.*, 61% meilleur que le plan de feux de Amiens et 78% que le plan de feux de SUMO.

La figure 4.10(c) montre que TAPIOCA génère également les plus petites files, tandis que la figure 4.10(d) montre la répartition des stratégies en fonction des usagers. Nous pouvons constater que TAPIOCA génère des temps d'attente plus courts pour 32% des usagers, et la version isolée pour 17% des usagers. Au total, la moitié des utilisateurs attendent moins avec notre algorithme, avec ou sans vagues vertes.

4. 6. 2 Scénario 2 : TAPASCologne

TAPASCologne[2] est l'un des plus grands – si ce n'est le plus grand – jeu de données à destination de SUMO. Il modélise la ville de Cologne (Köln, Allemagne) en se basant sur sa carte OpenStreetMap. Il intègre deux heures de trafic, c.-à-d. plus de 70 000 véhicules. Il est représenté sur la figure 4.12.

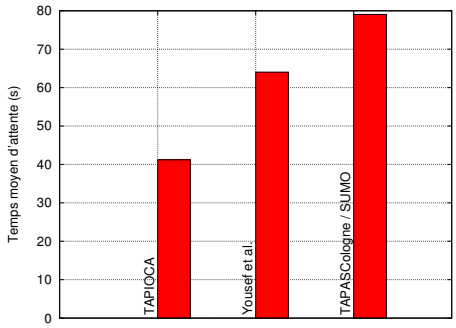
Les résultats sont présentés sur la figure 4.11 et montrent le temps moyen d'attente des usagers, ainsi que la file d'attente moyenne. Nous pouvons voir, sur les figures 4.11(a), 4.11(b) et 4.11(c), que TAPIOCA accompli toujours le meilleur temps d'attente, en comparaison à l'algorithme de Yousef *et al.* (réduction de 36%) et au plan de feux prédéterminé de TAPASCologne / SUMO (réduction de 48%). La figure 4.11(d) montre que TAPIOCA donne également la taille de file d'attente la plus petite, en moyenne. Finalement, les figures 4.11(e) et 4.11(f) montrent le temps moyen entre deux sélections successives du même mouvement : nous pouvons voir que TAPIOCA accompli en moyenne le temps le plus petit. Ces résultats nous prouvent notamment que la stratégie adoptée se comporte bien sur des scénarios à grande échelle et réalistes.

4. 6. 3 Scénario 3 : grilles d'intersections

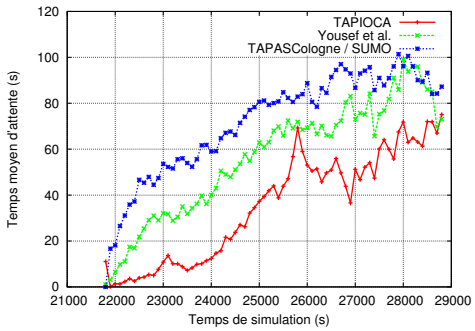
La figure 4.13 montre les performances de TAPIOCA sur une grille de 64×64 intersections (4 096 intersections). Chaque simulation dure 7 200 s et voit passer en moyenne 49 000 véhicules, avec un taux d'arrivée de $\lambda = 8$ véhicules par seconde. D'autres simulations réalisées sur 16, 64, 256, 1 024 et 4 096 intersections, avec une intensité de $\lambda = 0.5$, 1, 2, 4 et 8 sont disponibles en ligne[3]. Les trajets des véhicules ont été générés aléatoirement par un utilitaire fourni par SUMO.

Nous pouvons voir sur les figures 4.13(a), 4.13(b) et 4.13(c) une nette différence entre notre algorithme et les plans de feux prédéterminés de SUMO : notre algorithme créé des vagues vertes et réduit le temps moyen d'attente de 97%. Comparé à Yousef *et al.*, il réduit le temps moyen d'attente de 85%. Pour le cas de SUMO, cette différence est facilement explicable : les plans de feux étant prédéterminés, ils ne sont pas capables de s'adapter au trajet des véhicules, ce qui provoque des arrêts à chaque intersection, et donc une perte de temps. A titre d'exemple, la capture d'écran 4.14

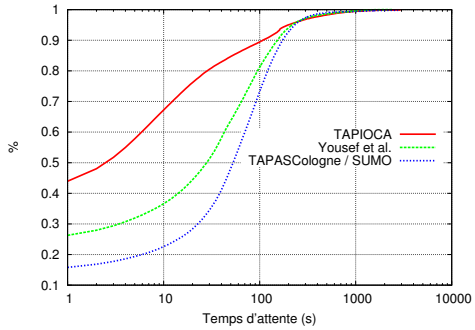
2. http://sumo.sourceforge.net/doc/current/docs/userdoc/Data/Scenarios/TAPASCologne.html
3. http://tapioca.sfaye.com/Grid/



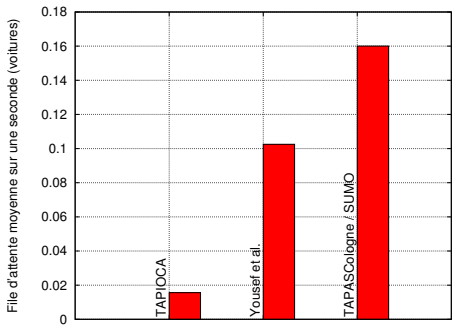
(a) Temps moyen d'attente.



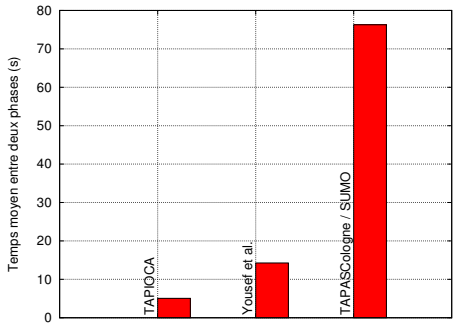
(b) Evolution du temps d'attente.



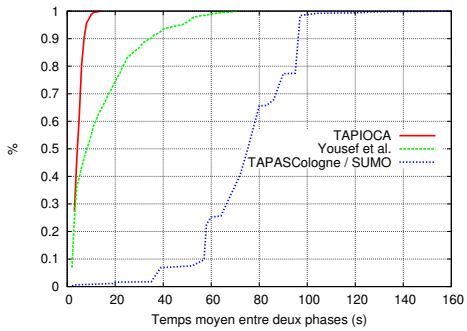
(c) CDF du temps d'attente (échelle logarithmique).



(d) Longueur de file moyenne.



(e) Intervalle moyen entre la sélection d'un mouvement.



(f) CDF de l'intervalle entre la sélection d'un mouvement.

FIGURE 4.11 – Comparaison de performances : le scénario TAPASCologne.

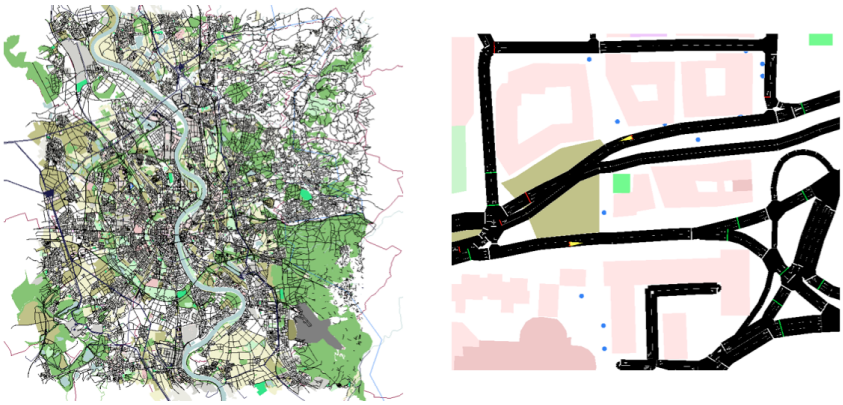
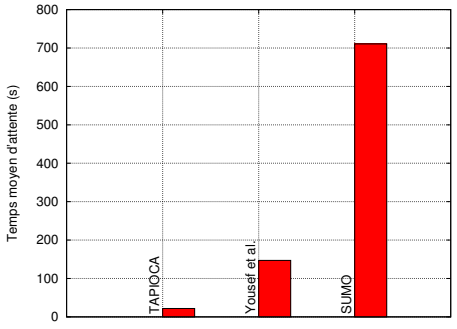
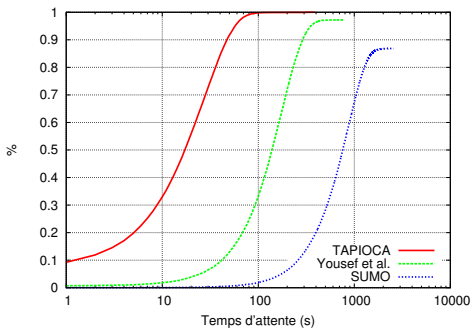


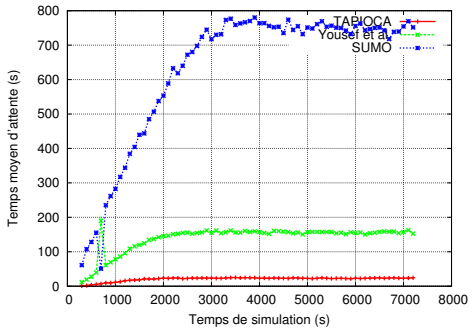
FIGURE 4.12 – Le scénario TAPASCologne.



(a) Temps moyen d'attente.



(b) CDF du temps d'attente (échelle logarithmique).



(c) Evolution du temps d'attente.

FIGURE 4.13 – Comparaison de performances : grilles d'intersections.

montre une simulation confrontant TAPIOCA à un plan de feux généré par SUMO. Le principe est simple : faire parcourir deux ensembles de véhicules d'un bout à l'autre de la grille d'intersections. Au final, les intersections gérées par TAPIOCA savent s'adapter aux arrivées de véhicules, ce qui n'est pas le cas pour un plan de feux statique, occasionnant des arrêts importants (véhicules rouges).

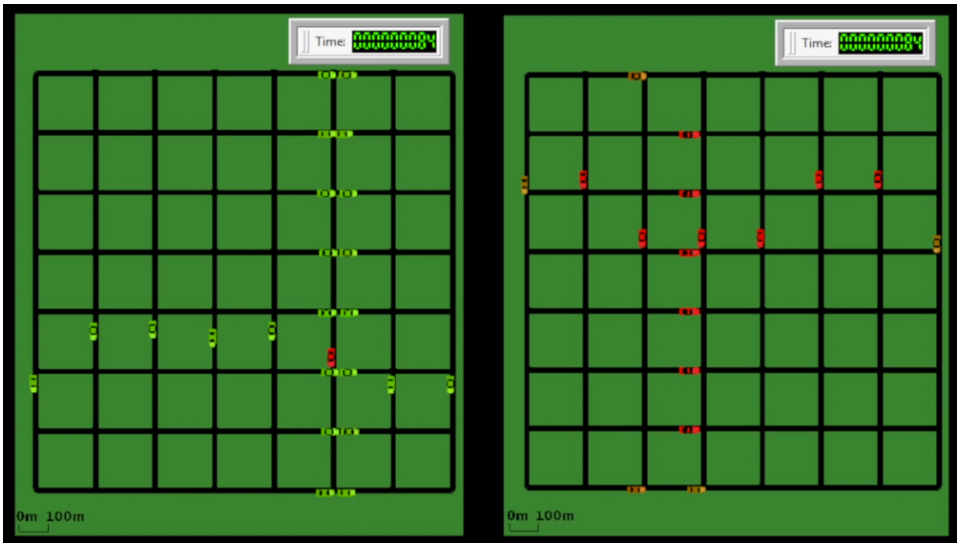


FIGURE 4.14 – Simulation confrontant TAPIOCA (à gauche) à des plans de feux statiques (à droite).

Pour le cas de Yousef *et al.*, nous pouvons l'expliquer par le fait que l'algorithme est conçu pour des temps de cycle trop longs. De plus, TAPIOCA se passe complètement de la notion de cycle et calcule les temps de phase juste avant leur application. Pour le cas de l'algorithme de Yousef *et al.*, cette détermination est faite au moment de la création du cycle : sur de grandes cartes à trafic moyen, où les apparitions et départs de véhicules sont courant, le délai entre une prise de décision et son application doit être court. Des arrêts sont donc provoqués, notamment à cause du manque de réaction de l'algorithme.

4. 7 Conclusion

Dans ce chapitre, nous avons proposé et évalué un algorithme adaptatif de contrôle des feux de circulation distribué sur plusieurs intersections et qui utilise un réseau de capteurs sans fil. Sur la base de trois scénarios différents, nous avons montré par simulation que cet algorithme était capable de réaliser un meilleur temps d'attente, temps de trajet et longueur de file d'attente par rapport à une solution prédéterminée, mais également par rapport à des solutions adaptatives. Outre les performances brutes de l'algorithme, ces résultats montrent qu'il existe un intérêt dans la gestion du trafic à l'échelle de l'intersection et de son voisinage direct. Ils nous montrent aussi qu'il est probablement suffisant de se limiter à une distance de communication à un saut (une intersection voisine), tout du moins dans le cas d'un scénario général. L'architecture distribuée permet de réagir rapidement à des situations de congestion

en prenant des décisions locales. Nous avons vu que la synchronisation d'intersections permettait d'améliorer les performances, mais il est probable que des messages puissent être économisées, par exemple lorsque le trafic est stable.

Notre algorithme part du principe que le réseau de communication est idéal. En réalité, la transmission sans fil est peu fiable, et peut entrainer des retards ou des pertes de paquets. Dans le chapitre suivant, nous nous intéressons à la manière dont se comporte TAPIOCA lorsque des informations manquent.