Algorithmes de coloriage dans les réseaux sans-fil à radio cognitive

Encadrant : **T. Nguyen**

Etudiants : **Y. Aouam, Y. Badis,**
J. Khalaf, S. Tiguemounine

Table des matières

1 Cahier des charges

Le projet intitulé « Algorithmes de coloriage dans les réseaux sans fil à radio cognitive » s'inscrit dans le cadre de l'UE PRES de la première année du master informatique parcours RES à la Faculté des Sciences et Ingenierie de Sorbonne Université. Ce projet est proposé par le professeur Trang Nguyen.

Entre l'émergence des services sans fil à haut débit qui nécessitent un accès rapide au spectre de fréquences radio d'une part et la limitation des ressources fréquentielles d'autre part, une nouvelle génération de réseau sans fil est devenu plus que nécessaire.

Dans la littérature, plusieurs propositions ont été faites, pour une meilleure utilisation du spectre de fréquence. La radio cognitive est l'une des technologies les plus prometteuses.

La radio cognitive est une technologie qui a émergé a la fin des années 90 qui permet aux utilisateurs des bandes libres, appelés utilisateurs secondaires, d'exploiter le spectre de fréquences alloué aux utilisateurs des bandes licenciées, appelés utilisateurs principaux, de manière opportuniste

Ce problème d'allocation intelligente et dynamique des ressources radio peut être traité par un outil mathématique appelé « coloriage des graphes ». En effet, plusieurs algorithmes existent dans ce cadre et présentent des performances variées.

Le projet consiste à étudier le comportement de trois algorithmes de coloriage dans un réseau Ad-hoc sans-fil à radio cognitive : DSATUR, coloriage aléatoire et coloriage distribué à vue locale. L'étude porte en particulier sur une comparaison des performances de ces algorithmes dans différents modèles de mobilité, différentes topologies et différents modèles de trafic. À l'issue du projet, une recommandation sur les algorithmes appropriés dans chaque contexte sera faite.

2 Plan de développement

Durant le premier semestre, nous avions fixé, dans un premier temps, les attentes et l'objectif du projet. Nous avions commencé notre projet par une synthèse sur les réseaux à radio cognitive et les algorithmes de coloriage grâce à une recherche bibliographique dans laquelle nous avions amassé des articles et des livres électroniques et papiers, puis nous avions travaillé sur les plus pertinents.

Nous avions ensuite étudié le comportement de l'algorithme DSATUR qui est un algorithme de coloriage de sommets basé sur le concept de degré de saturation, puis nous avions définis une solution envisagée pour la suite de notre projet.

Dans ce deuxième semestre, nous avons réalisé la suite de notre projet en plusieurs étapes, à travers des réunions hebdomadaires sous la supervision de notre encadrant.

Nous avons commencé par la conception d'un algorithme distribué de coloriage de sommets de type DSATUR, nommé DSATUR_DISTIBUE, en mettant en place un pseudo-code et en essayant de l'adapter à différents cas de figure pour avoir une solution cohérente. Nous avons également conçu une approche algorithmique distribuée pour le coloriage d'arêtes basée sur le degré de saturation (DSAT). Cette approche, nommé Edge_DSATUR, est présentée uniquement dans l'annexe A.

Nous avons ensuite consacré une semaine pour étudier la documentation officielle d'OMNET++ afin de nous familiariser avec l'outil qui nous a servi par la suite comme simulateur pour implémenter nos algorithmes. Par la suite, nous avons implémenté les deux algorithmes DSATUR_DISTIBUE et Edge_DSATUR sur OMNET++ et vérifié leurs exactitude en les testant sur différentes topologies.

Après avoir finalisé l'algorithme DSATUR_DISTIBUE, mentionné dans le cahier des charges, nous avons enchainé avec un second algorithme de coloriage d'arêtes à vue locale. Nous avons écrit un pseudo-code et l'avons déroulé sur différentes topologies pour traiter tous les cas possibles. Ensuite, nous l'avons implémenté sur OMNET++ afin de visualiser son comportement.

Le cahier des charges qui nous a été parvenu indique un troisième algorithme de coloriage aléatoire d'arêtes, que nous avons également implémenté sur le simulateur.

Les trois algorithmes étant implémentés, nous avons donc optimisé leurs codes avant d'étudier leurs performances.

Afin de comparer les performances de nos algorithmes, nous avons établi une relation entre le graphe sur lequel nous réalisons un coloriage d'arêtes avec son graphe correspondant pour un coloriage de sommets.

Dans la phase d'études des performances, nous avons exécuté plusieurs simulations et tracé les courbes de performances de chaque algorithme selon 3 critères. Enfin, nous avons fait une analyse sur les résultats obtenus en visualisant les courbes résultantes.

D'une manière plus claire, nous énumérons les tâches réalisées ci-dessous :

- Conception d'un algorithme distribué de coloriage de sommet, DSATUR_DISTIBUE.
- Conception d'un algorithme distribué de coloriage d'arêtes, Edge_DSATUR.
- Etude de la documentation officielle d'OMNET++.
- Implémentation des algorithmes DSATUR_DISTRIBUE et Edge_DSATUR.
- Conception de l'algorithme de coloriage d'arêtes à vue locale.
- Implémentation de l'algorithme de coloriage d'arêtes à vue locale.
- Implémentation de l'algorithme de coloriage aléatoire d'arêtes.
- Etude des performances des algorithmes proposés.
- Rédaction du rapport.

Pour une meilleure visualisation du séquencement des tâches réalisées, nous utilisons le diagramme de Gantt. La Figure 2.1 représente le diagramme de Gantt du semestre 2.

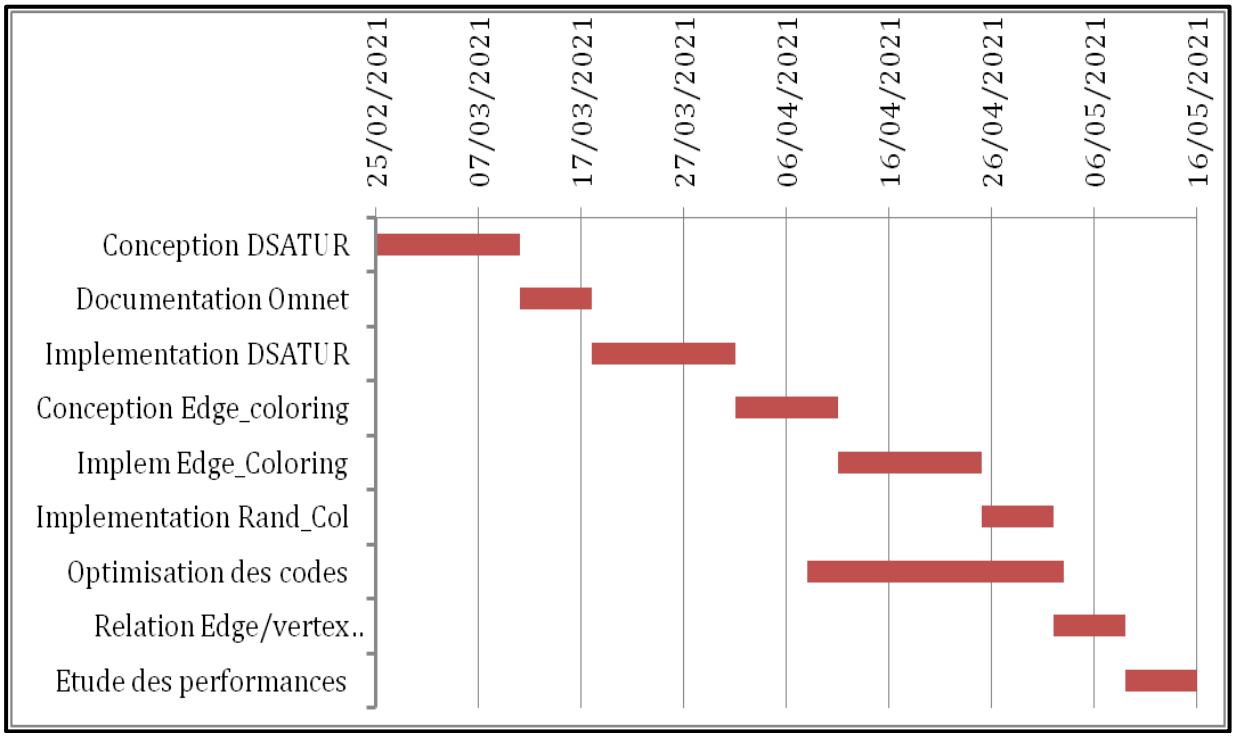

Figure 2.1: Diagramme de Gantt.

3 Bibliographie

3.1 Réseau à radio cognitives

3.1.1 Introduction

Entre l'émergence des services sans fil à haut débit qui nécessitent un accès rapide au spectre de fréquences radio et le problème des fréquences radio qui représente une ressource naturelle limitée, une nouvelle génération de réseaux sans fil est devenue plus que nécessaire. Dans la littérature, plusieurs propositions ont été faites, pour une meilleure utilisation du spectre de fréquences. La radio cognitive est l'une des technologies les plus prometteuses. [1]

3.1.2 La technologie radio cognitive

La radio cognitive (CR – Cognitive Radio) est une technologie qui permet aux utilisateurs des bandes libres, appelés utilisateurs secondaires, d'exploiter le spectre de fréquences alloué aux utilisateurs des bandes licenciées, appelés utilisateurs principaux, de manière opportuniste.

Ainsi, elle résout les problèmes de pénurie du spectre de fréquences dans les communications sans fil dont souffrent les réseaux sans fil actuels. [2]

CR correspond à un système de radiocommunication intelligent qui a la connaissance de son environnement et qui y détecte tout changement. [1]

3.1.3 Architecture des réseaux à radio cognitive

Un réseau à radio cognitive (CRN – Cognitive Radio Network), également appelé réseau secondaire, se compose d'un certain nombre de réseaux radio primaires (PRN – Primary Radio Network).

Un réseau primaire est un réseau utilisant des bandes sous licence, dont les utilisateurs accèdent uniquement au spectre de fréquence alloué à ce réseau. Les utilisateurs secondaires (SU – Secondary User) exploitent le spectre sans licence, mais ils peuvent également avoir accès au spectre sous licence disponible de tous les PRNs composant le CRN, d'une manière opportuniste et sous certaines contraintes imposées par les PRNs.

Les transmissions des utilisateurs primaires ne sont pas affectées par les utilisateurs secondaires car les utilisateurs primaires (PU – Primary User) ont la priorité en ce qui concerne l'accès au spectre qui leur est dédié.

Les CRNs et les PRNs, peuvent être des réseaux centralisés avec infrastructure ou distribués en mode ad-hoc. [3]

3.1.4 Réseau à radio cognitive centralisé

Un réseau à radio cognitive centralisé est basé sur une infrastructure dans laquelle les stations de bases à radio cognitive (CR BTS – Cognitive Radio Base Transceiver Station) contrôlent et coordonnent les transmissions des utilisateurs secondaires dans les bandes de fréquences sous licence et sans licences à la fois. La Figure 3.1 illustre un réseau à radio cognitive centralisé.

En effet, avec les informations relatives au spectre de fréquences, collectées auprès des utilisateurs à radio cognitive, les stations de bases prennent les décisions d'accès au spectre pour tous les nœuds du réseau. [3]

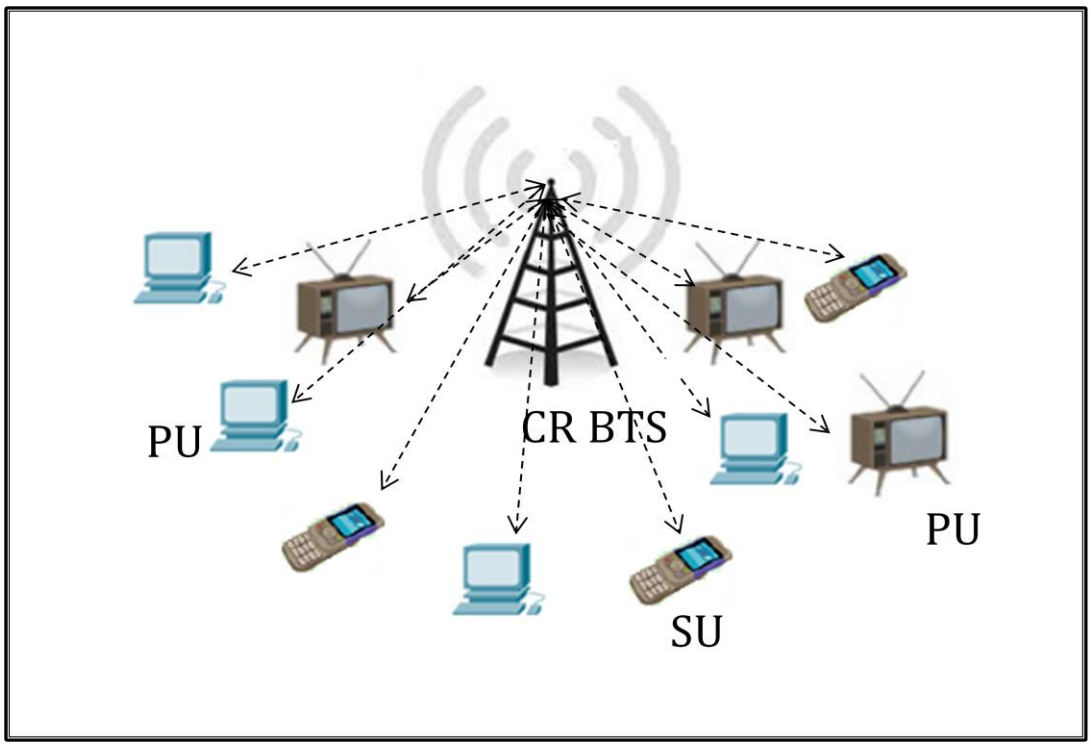

Figure 3.1: Réseau à radio cognitive avec infrastructure.

La norme IEEE 802.22 est la première norme mondiale pour les réseaux à radio cognitive centralisés basés sur une infrastructure.
Un réseau IEEE 802.22 est un réseau régional sans fil (WRAN –Wireless Regional Area Networks) qui définit les spécifications des communications des utilisateurs à radio cognitive sur les bandes de fréquences non utilisées de la télévision. Le Tableau 3.1 regroupe les spécifications de la norme IEEE 802.22.
La station de base gère les utilisateurs à radio cognitive dans un rayon de 33 km. [4]

Paramètres	Spécifications
Rayon de cellule	30-100 km
Méthodologie	Détection de spectre
Modulation	OFDMA
Capacité du canal	18Mbps
Capacité utilisateur	1.5 Mbps (uplink) 384 Kbps (downlink)

Tableau 3.1: Spécifications de la norme IEEE 802.22.

3.1.5 Réseau à radio cognitive distribué

Dans un réseau à radio cognitive distribué, les utilisateurs à radio cognitive communiquent entre eux via des connexions ad hoc point à point sur les bandes de fréquences sous licence ou sans licence. La Figure 3.2 illustre un réseau à radio cognitive distribué.

Un tel réseau bénéficie d'une réduction avantageuse du coût de l'infrastructure, néanmoins l'augmentation de la complexité du réseau dû à l'absence d'une infrastructure est conséquente.

En effet, en l'absence d'un contrôle centralisé, les utilisateurs radio cognitifs coordonnent leurs décisions d'accès au spectre de fréquence pour partager les opportunités spectrales disponibles. Le mécanisme de synchronisation est nécessaire pour une telle coordination. Aussi, des techniques de détection et de communication coopératives distribuées sont utilisées pour améliorer les performances du réseau.

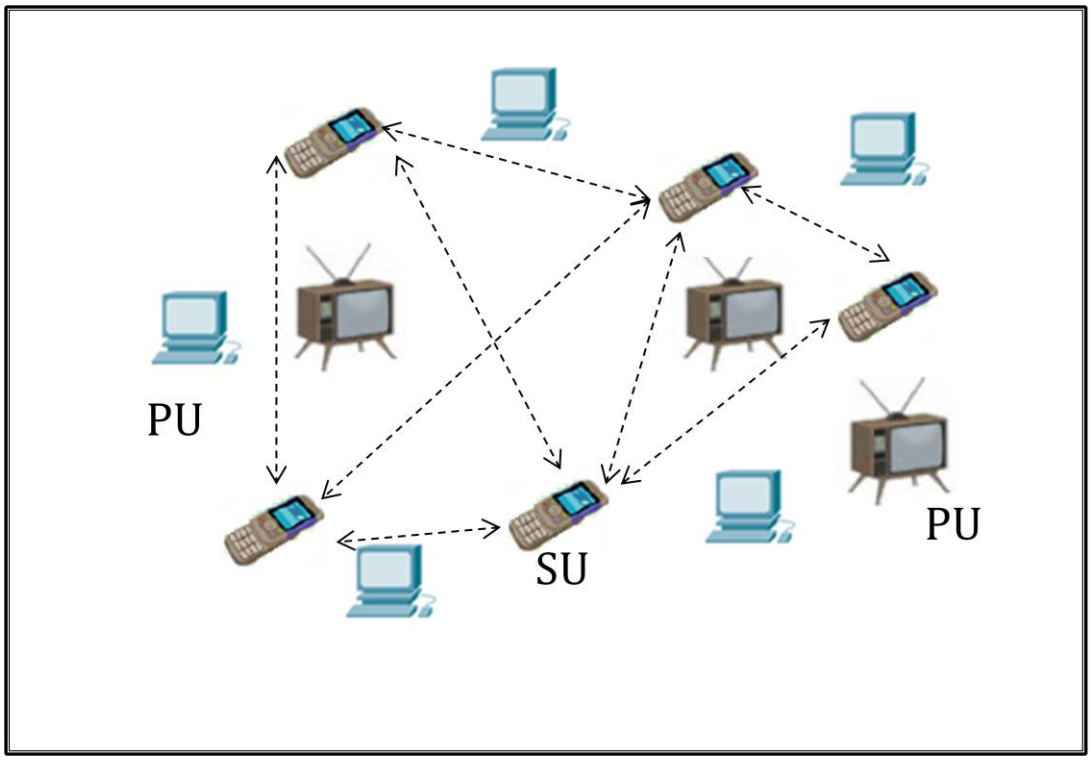

Figure 3.2: Réseau ad hoc à radio cognitive distribué.

3.1.6 L'accès opportuniste au spectre dans les réseaux à radio cognitive

L'accès opportuniste au spectre (OSA – Opportunistic Spectrum Access) fait référence au fait que les utilisateurs secondaires exploitent dynamiquement les bandes de fréquences non utilisées par des utilisateurs primaires. La non-utilisation du spectre par les utilisateurs primaires est appelée une opportunité spectrale, un trou spectral ou un espace blanc.

OSA veille à ce que les utilisateurs secondaires ne perturbent pas les transmissions des utilisateurs primaires. Aussi, il restitue l'accès au spectre aux PUs lorsqu'ils redeviennent actifs. La Figure 3.3 illustre le concept d'accès dynamique au spectre de fréquences. [3]

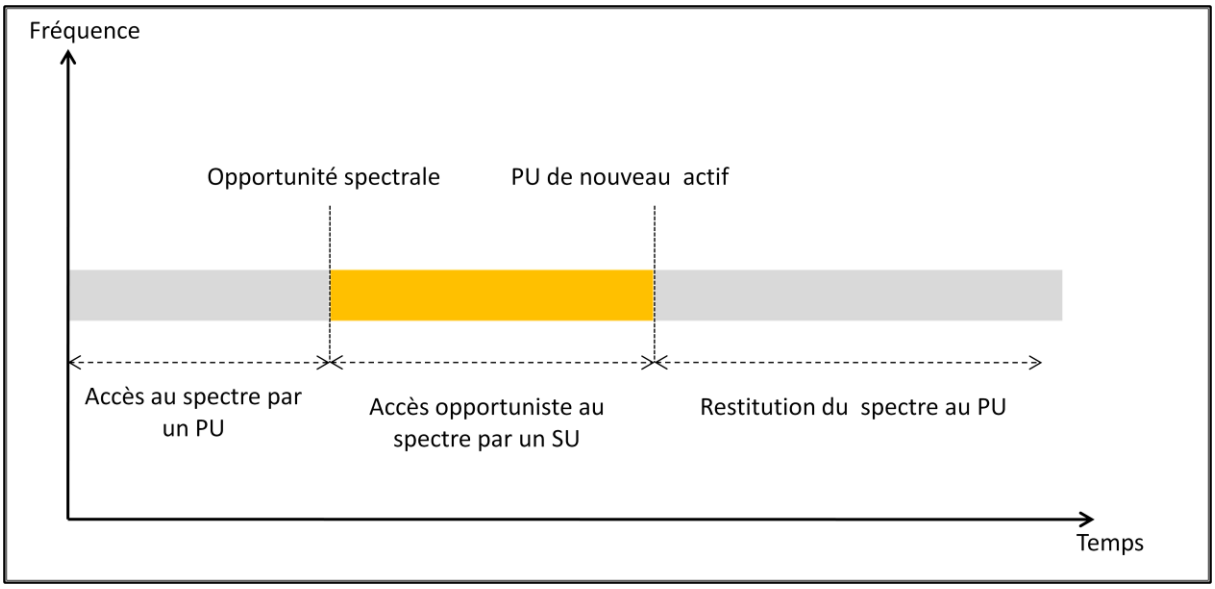

Figure 3.3: L'accès opportuniste au spectre de fréquences.

3.1.7 Capacité cognitive d'une radio cognitive

La capacité cognitive d'une radio cognitive est définie comme étant la capacité de l'émetteur-récepteur radio cognitif à détecter l'environnement radio environnant, à analyser les informations capturées et à décider en conséquence, de la bande de spectre à utiliser et de la meilleure stratégie de transmission à adopter. La figure 3.4 illustre le cycle des principales fonctions qui régissent la radio cognitive.

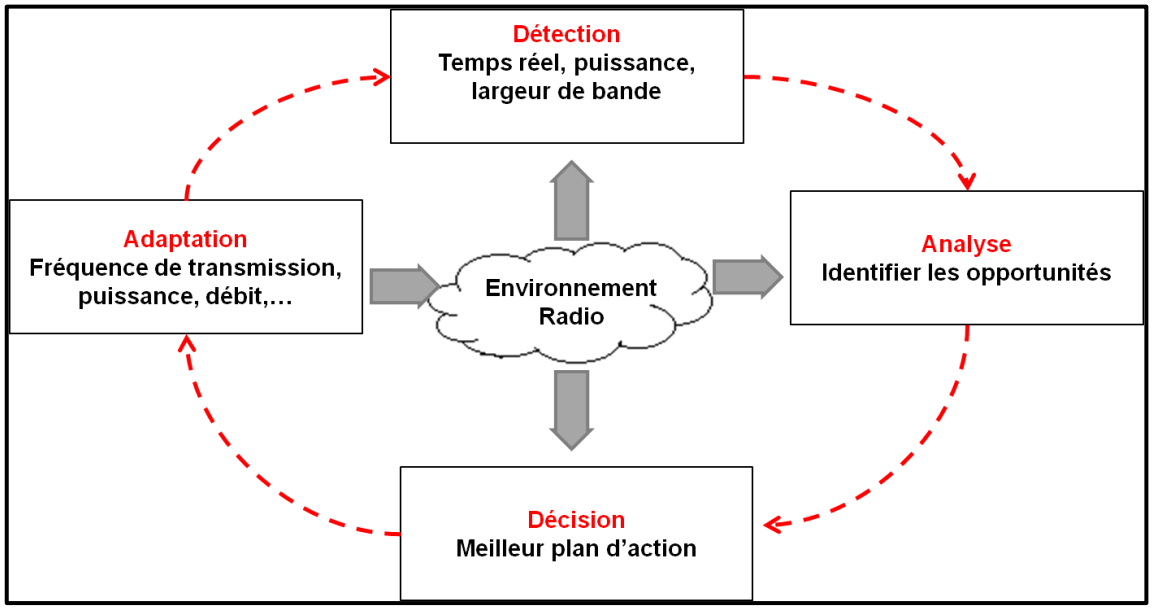

Figure 3.4: Architecture fonctionnelle d'une radio cognitive.

3.1.7.1 Détection du spectre

La détection du spectre fait référence à la capacité d'une radio cognitive à mesurer les activités électromagnétiques, dues aux transmissions radio en cours, sur différentes bandes du spectre. Ainsi, des paramètres liés à ces bandes, tel que les niveaux de puissance et l'activité des utilisateurs, sont capturés. Les informations du spectre détectées doivent être suffisantes pour que la radio cognitive parvienne à des conclusions précises concernant l'environnement radio.

La détection du spectre est l'une des fonctions les plus critiques d'une radio cognitive. En effet, la radio cognitive doit prendre des décisions en temps réel sur les bandes à détecter. Ainsi, la détection du spectre doit être rapide afin de suivre les variations temporelles de l'environnement radio. [3]

3.1.7.2 Analyse du spectre

L'analyse spectrale consiste à identifier l'existence d'opportunités spectrales dans l'environnement radio environnant sur la base des paramètres d'environnement radio détectés. [3]

3.1.7.3 Décision relatives à l'accès au spectre

La dernière étape du cycle de cognition d'une radio cognitive consiste à décider de l'ensemble des actions de transmission à entreprendre en fonction du résultat des procédures de détection et d'analyse du spectre. L'ensemble des actions inclut le spectre le plus favorable pour une transmission à venir, l'instant où une transmission sur une certaine bande doit commencer, la puissance de transmission maximale, la vitesse de modulation et bien d'autres. [3]

3.1.8 L'auto-configuration de la radio cognitive

La radio cognitive est capable de s'auto-configurer dynamiquement en ajustant ses paramètres de transmission en temps réel sans modification des composantes matérielles. Cette caractéristique permet à la radio cognitive de s'ajuster par rapport à l'environnement dynamique où elle opère. Dans ce qui suit certains des paramètres qui doivent être reconfigurables. [3]

3.1.8.1 Fréquence d'opération

La radio cognitive est capable de changer de fréquence d'opération quand cette dernière devient indisponible. Le choix de la nouvelle fréquence est basé sur l'information collectée de l'environnement radio. La fréquence la plus appropriée est sélectionnée et la communication est réalisée sur cette nouvelle fréquence. [3]

3.1.8.2 Modulation

La radio cognitive reconfigure son schéma de modulation selon les besoins de l'utilisateur et selon la bande spectrale. Dans le cas des applications telles que la voix sur IP, la radio cognitive doit sélectionner le schéma de modulation qui garantit la meilleure

efficacité spectrale, le débit le plus élevé. Dans le cas des applications qui ne supportent pas un taux d'erreur élevé, la radio cognitive doit sélectionner le schéma de modulation qui garantit le taux d'erreur le plus bas.

3.1.8.3 Puissance de transmission

La nouvelle puissance de transmission est choisie selon les nouvelles contraintes que la radio cognitive doit respecter afin d'utiliser les bandes spectrales d'une manière opportuniste. Le contrôleur de puissance est l'entité qui rend possible cet ajustement de puissance. Il assure que les seuils d'interférence ne soient pas dépassés dans l'environnement radio.

3.1.8.4 Technologie de communication

L'existence de plusieurs types de réseaux dans une zone géographique, conduit à la coexistence de plusieurs technologies de communication. Un des aspects de la radio cognitive est l'interopérabilité.

3.1.9 Applications des réseaux à radio cognitive

Le réseau à radio cognitive et l'accès opportuniste au spectre de fréquence sont utilisés dans différentes applications.

3.1.9.1 Réseaux de maillage à radio cognitive

Les réseaux maillés sans fils traditionnels sont confrontés à la rareté de la bande passante nécessaire pour répondre aux exigences des applications haut débit existantes.
L'accès opportuniste au spectre de fréquence résout le problème de pénurie de la bande passante des réseaux maillés en permettant aux nœuds du réseau d'exploiter dynamiquement les opportunités spectrales disponibles. [3]

3.1.9.2 Réseaux de sécurité publique

Les réseaux de sécurité publique sont utilisés pour les communications entre les policiers, les pompiers et le personnel paramédical. Ils exploitent la radio cognitive pour faire face la quantité limitée du spectre alloué. [3]

3.1.10 Conclusion

Dans cette partie, nous pouvons déjà déduire que la radio cognitive peut résoudre le problème de pénurie des fréquences. Dans la partie suivante, nous nous intéressons aux algorithmes de coloriages.

3.2 Les algorithmes de coloriage

3.2.1 Introduction

Les graphes peuvent être utilisés pour modéliser un grand nombre de domaines problématiques tels que les réseaux sociaux, la navigation par satellites, la planification et les réseaux informatiques.

3.2.2 La coloration de graphes

Le problème de coloration de graphe est l'un des problèmes les plus connus dans le domaine de la théorie des graphes. Ce problème se résume à comment attribuer des couleurs à tous les sommets de n'importe quel graphe de sorte que deux sommets joints par une arête n'ait pas la même couleur et que le nombre de couleurs utilisées pour colorier le graphe soit minime. [5]

3.2.3 Vertex coloring

Vertex coloring, coloration de sommets en français, consiste à affecter des couleurs à tous les sommets d'un graphe G de sorte à vérifier deux conditions :

_ Deux sommets partageant la même arête, ne doivent pas avoir la même couleur.

_ Le nombre de couleurs utilisées doit être minimal. [6]

3.2.4 Edge coloring

Edge coloring, coloration des arêtes en français, consiste à affecter des couleurs à toutes les arêtes d'un graphe G de sorte à vérifier deux conditions :

_ Deux arêtes partageant le même sommet, ne doivent pas avoir la même couleur.

_ Le nombre de couleurs utilisées doit être minimal. [5]

3.2.5 Définitions

Afin de poursuivre, il est nécessaire de définir quelques terminologies liées à la théorie des graphes.

_ Définition 1 : un graphe G= (V, E) est défini par un ensemble de sommets V de n sommets et un ensemble d'arêtes E de m arêtes

_ Définition 2 : si $\{u,v\} \in E$ alors les sommets u et v sont dits adjacents. On dit aussi que les sommets u et v sont incidents de l'arête $\{u,v\} \in E$.

_ Définition 3 : L'ensemble de voisins d'un sommet v, noté $\Gamma G(v)$ est l'ensemble des sommets adjacents à v dans le graphe G. On écrit : $\Gamma G(v) = \{u \in V : \{v,u\} \in E$. Le degré d'un sommet v est la cardinalité de son ensemble de voisins, noté $| \Gamma G(v) |$ ou plus généralement degG (v).

_ Définition 4 : Le nombre minimum de couleurs nécessaire pour colorer les arêtes d'un graphe est appelé indice chromatique.

_ Définition 5 : soit c (v)= NULL pour tout sommet v ∈ V non assigné à une classe de couleur. Le degré de saturation de v, noté sat (v), est le nombre de couleurs différentes attribuées aux sommets adjacents. On écrit : sat (v) = | {c(u) : u ∈ Γ (v) ∧ c(u) ≠ NULL} |. [5]

3.2.6 Algorithmes de coloriage de graphes

Il existe dans la littérature de nombreux algorithmes de coloriage de graphes, dans cette partie nous en présentons trois.

3.2.6.1 Algorithme GREEDY

L'algorithme GREEDY est l'un des algorithmes heuristiques les plus simples mais aussi le plus fondamental pour la coloration des graphes. L'algorithme prend les sommets du graphe un par un selon un certain ordre, éventuellement arbitraire, et attribue à chaque sommet la première couleur disponible. S'agissant d'un algorithme heuristique, la solution produite peut être non optimale.

En conséquence, plusieurs algorithmes de coloration de graphe qui cherchent à trouver un ordre de sommet correct, c'est-à-dire un ordre qui conduirait à une solution optimale quel que soit le graphe, ont été proposés.

3.2.6.2 Algorithme Welch-Powell

L'algorithme Welch-Powell est un algorithme de coloration de sommets qui ne propose pas toujours une solution optimale mais qui propose une meilleure solution qu'une simple coloration de sommets sans ordre défini.
L'algorithme Welch-Powell est définit comme suit :
1_ Trouver le degré de chaque sommet.
2_ Lister les sommets par ordre décroissant de leur degré.
3_ Attribuer la première couleur au premier sommet de la liste.
4_ Suivre la liste en attribuant cette même couleur à tout sommet non coloré et non adjacent à un sommet coloré de cette couleur.
5_ Si tous les sommets ne sont pas colorés, choisir une couleur qui n'est pas encore utilisée et recommencer les étapes 3 et 4. [6]

3.2.6.3 Algorithme DSATUR

L'algorithme DSATUR (abréviation de degré de saturation) a été proposé en 1979 par Brélaz. Il est similaire à l'algorithme GREEDY dans le sens où il colorie chaque sommet à la fois selon un certain ordre. Avec l'algorithme GREEDY, l'ordre est établi avant le début de la coloration, c'est là que réside la différence entre les deux algorithmes GREEDY et DSATUR. En effet, avec l'algorithme DSATUR, le choix du sommet à colorer est décidé de manière heuristique en fonction de la coloration partielle du graphe. Ce choix est basé sur le degré de saturation des sommets (voir la définition 5). [5]

L'algorithme DSATUR est défini comme suit :

1_ Lister les sommets par ordre décroissant de leur degré.

2_ Attribuer la première couleur à un sommet de degré maximum.

3_ Choisir un sommet de degré de saturation maximal. En cas d'égalité, choisir un sommet de degré maximal.

4_ Attribuer la plus petite couleur possible à ce sommet.

5_ Si tous les sommets ne sont pas colorés, aller à l'étape 3.

3.2.7 Vertex to edge coloring

Le problème de coloration des arêtes et le problème de coloration des sommets sont très étroitement liés. En effet, il est possible de colorer les arêtes de n'importe quel graphe en colorant simplement les sommets du graphe linéaire correspondant.

Étant donné un graphe G, le graphe linéaire de G, noté L (G), est construit de la manière suivante :

_ Chaque arête de G devient un sommet dans L (G)

_ Connecter des paires de sommets dans L (G) si et seulement si les arêtes correspondantes dans G partagent un sommet commun comme extrémité. [5]

Tel que :

$$Graphe\ G: \begin{cases} n\ sommets \\ m\ arêtes \end{cases}$$

Et :

$$Graphe\ L(G): \begin{cases} m\ sommets \\ \frac{1}{2}\sum_{v \in V} \deg(v)^2 - m\ arêtes \end{cases}$$

Un exemple de conversion d'un graphe G à un graphe linéaire L (G) est illustré dans la Figure 3.5.

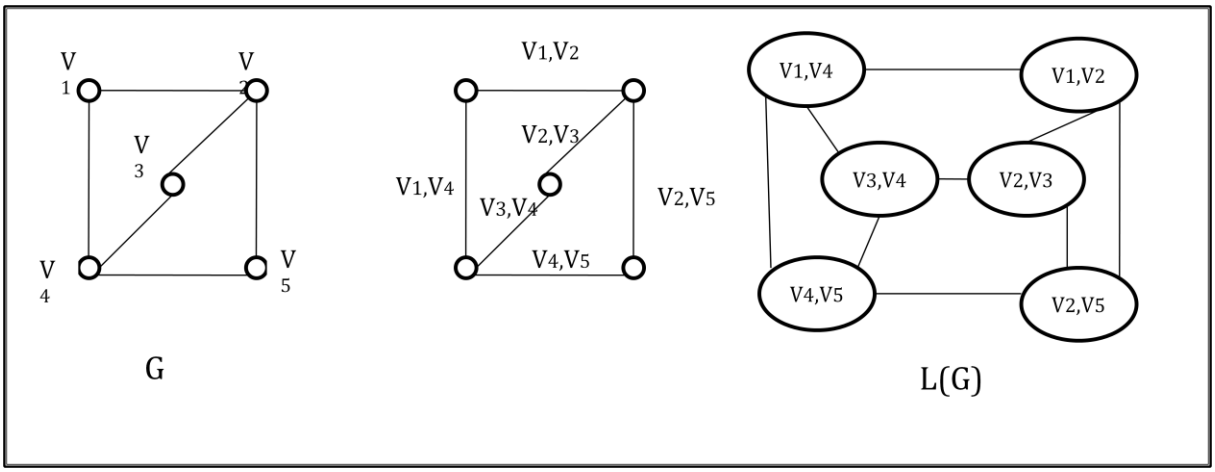

Figure 3.4: Conversion du graphe G en son graphe linéaire L(G).

3.2.8 Conclusion

Dans cette partie, nous nous sommes familiarisés avec le coloriage d'algorithme. Dans la partie suivante, nous nous intéressons à l'analyse du problème de coloriage d'un point de vue de la radio cognitive.

4 Analyse

4.1 Analyse du problème de coloriage d'un point de vue de la radio cognitive

Dans un réseau ad-hoc à radio cognitive, le problème d'attribution des fréquences peut être modélisé par un coloriage de graphe dont les sommets représentent les nœuds du réseau, les arêtes représentent les canaux de communications entre les utilisateurs et l'ensemble des couleurs représente les fréquences porteuses.
Un réseau ad-hoc est un réseau sans fil décentralisé car il ne s'appuie pas sur une infrastructure préexistante. Dans le graphe associé, chaque nœud a une portée radio et est dit connecté à un autre nœud si ce nœud dit voisin est à sa portée radio, la connectivité entre ces deux nœuds est représentée par une arête entre les deux sommets associés.

La première problématique rencontrée dans une telle situation, vient du fait qu'aucun nœud n'a une vision globale du réseau, ce qui rend la coloration du graphe de manière la plus optimale possible une tâche très complexe. Cette complexité est accentuée à cause de la mobilité des nœuds et des obstacles dans le réseau, par conséquent, la coloration doit prendre en considération le caractère dynamique du graphe.

La modélisation adéquate au problème de l'allocation des fréquences dans un réseau à radio cognitive est un coloriage des arêtes du graphe et non un coloriage de sommets car les fréquences porteuses qui sont représentées par l'ensemble des couleurs s'appliquent sur les canaux radio qui sont modélisés par les arêtes du graphe.

Afin de réaliser un coloriage performant, notre algorithme doit prendre en considération le modèle de mobilité pour être optimisé pour une situation précise. Un réseau où les utilisateurs se déplacent rapidement présente plus de contrainte en terme de temps de calculs dans les nœuds et en terme de quantité de messages d'informations échangés. Ainsi, une information communiquée par un nœud à un autre à un instant t, pourrait ne pas être valable à un instant $t+t_0$.

Nous constatons également qu'en pratique, le nombre de fréquences disponibles est très limité et par conséquent, la coloration avec un nombre de couleurs donné est parfois impossible, cette contrainte nous impose de colorier dans certains cas deux arêtes incidentes au même sommet avec la même couleur, ce qui est interprété par une attribution de la même fréquence pour deux liens de communications qui peuvent s'interférer. Cette contrainte peut causer une diminution des performances du réseau et sera prise en considération dans la solution proposée pour trouver un compromis entre un nombre chromatique minimal et une valeur suffisamment élevée du SIR (Signal-to-Interference Ratio).

4.2 Vers un algorithme de coloriage distribué à vue locale

La plupart des algorithmes de coloriage existants, dont l'algorithme DSATUR, ont une approche centralisée et supposent qu'il existe au moins un nœud possédant une vue globale sur le réseau [7]. Tandis que dans un réseau sans-fil à radio cognitive il est impossible qu'un nœud puisse avoir une vision globale sur le réseau, il est donc primordial d'adapter l'algorithme de coloriage en un algorithme distribué à vue locale dans lequel chaque nœud établit ses propres calculs et communique ses résultats avec les autres nœuds existant par échange de messages.

L'échange d'informations entre les nœuds se fait par des messages de contrôle indiquant les fréquences disponibles détectées par le nœuds lui-même ainsi que par ses voisins, la fréquence choisie, le taux d'interférences accepté et un message indiquant que l'utilisateur primaire s'est présenté.

Lors de la conception d'un algorithme à vue locale, le choix du nombre de sauts effectués par un nœud pour découvrir les autres nœuds du réseau est un facteur très important. Un nœud ne peut pas utiliser la même fréquence pour deux communications différentes, c'est la raison pour laquelle il doit collecter les informations de ses voisins avant de choisir d'exploiter la fréquence disponible.

5 Conception

Dans ce projet, nous avons conçu trois algorithmes distribués : un algorithme de coloriage de nœuds basé sur l'approche du DSATUR, un algorithme de coloriage d'arêtes à vue locale et un algorithme de coloriage d'arêtes basé sur l'approche du DSATUR (présenté dans l'annexe A). Les deux premiers algorithmes ainsi que l'algorithme aléatoire de coloriage d'arêtes ont été ensuite implémentés pour comparer leurs performances.

5.1 Conception de l'algorithme DSATUR_DISTRIBUE

Dans la description de l'algorithme DSATUR centralisé, nous avons vu que l'intérêt principal de cet algorithme réside dans l'ordre de coloriage des sommets puisqu'il commence par colorier les sommets ayant le plus grand degré de saturation (le plus grand nombre de couleurs différentes attribuées aux sommets adjacents), ce qui signifie que les nœuds qui présentent un choix minimal, plus de contraintes, de couleurs disponibles sont traités en priorité.

Cependant, dans un réseau ad-hoc sans fil, aucun nœud n'a une vision globale du réseau, ce qui signifie que l'information sur le plus grand degré de saturation et le plus grand degré du réseau n'est détenue par aucun nœud.

Nous considérons, dans un premier temps, que l'algorithme que nous proposons fonctionne dans un réseau à ID uniques. Les variables utilisées sont : **ID**, le degré **D**, le degré de saturation **DS**, un booléen qui indique si le nœud est un initiateur nommé **START**, la liste des voisins actifs **actif_neighbors**, la liste des couleurs disponibles **list_color**, la liste des nœuds restant à colorier **remain**, l'ID du nœud qui donne l'ordre de se colorier **master**, l'état du nœud s'il est déjà colorié nommée **colored**, la liste des nœuds coloriés **colored_set**, un dictionnaire qui stocke les degrés des voisins **neighbors_D** et un autre qui stocke les degrés de saturation des voisins **neighbors_DS**.
La première étape consiste à échanger des messages <**my_id**> entre un nœud et ses voisins, chaque nœud informe ses voisins de son ID ainsi que son degré et reçoit la liste des ID ainsi que les degrés de ses voisins. A l'issue de cette étape, chaque nœud sait s'il a le plus grand ID par rapport à ses voisins.

Après avoir reçu les identifiants de tous ses voisins, un nœud considère qu'il est initiateur s'il a le plus grand degré (ou le plus grand id en cas d'égalité de degrés) et met la variable START = true (plus d'un nœud non voisins entre eux peuvent être initiateurs), prend la plus petite couleur dans sa palette et informe tous ses voisins à travers un message <my_color>.

Du fait que le coloriage n'a pas encore commencé pour les autres nœuds, la notion du degré de saturation n'a pas de signification car tous les nœuds voisins de l'initiateur ont un DS égal à 1. L'initiateur va donc regarder dans la liste des degrés de ses voisins et ordonne le voisin qui a le plus grand degré de prendre une couleur à travers le message <color_yourself>, ce message contient également la liste des nœuds restants non coloriés du nœud émetteur. Ce voisin auquel il a envoyé le message sera aussi supprimé de la liste des voisins actifs.

A la réception du message <my_color>, un nœud élimine la couleur contenue dans le message de sa liste des couleurs disponibles pour ne pas l'utiliser par la suite car elle a déjà été utilisée par son voisin, élimine l'émetteur du message de sa liste de voisins actifs pour ne pas le considérer dans la continuation du coloriage, ajoute l'émetteur à la liste des nœuds coloriés et incrémente la variable du degré de saturation DS.

A la réception du message <color_yourself>, un nœud marque d'abord l'émetteur du message comme master, prend la plus petite couleur disponible, met à jour sa variable **colored** en la mettant true, informe ses voisins en envoyant un message <my_color> et ajoute les éléments de la liste **remain** contenue dans le message <color_yourself> avec sa liste **remain** qu'il stocke s'ils ne font pas partie de sa liste **colored_set**. Il envoie ensuite un message <ds_request> pour demander les degrés de saturation de ses voisins actifs (non coloriés). Cependant, si la liste des voisins actifs est vide et que la liste des nœuds restant **remain** n'est pas vide, alors le nœud considère qu'il reste encore des nœuds non coloriés dans le graphe et renvoie donc à son master un message <impass>.

A la réception du message <ds_request>, si le nœud n'est pas encore colorié (colored=false) répond par un message <ds_response> contenant le degré de saturation. Si le

nœud est déjà colorié, il répond par un message <already_colored>. Ce cas de figure peut apparaitre si un voisin d'un nœud colorié lui envoie un message <ds_request> avant que son message <my_color> ne soit reçu par ce voisin (pour simplifier le pseudo code donné un peu plus loin, <already_colored> n'est pas décrit bien qu'il soit considéré dans le programme d'implémentation sur le simulateur), voir annexe B.

.

A la réception des messages <ds_response> envoyés par tous les voisins actifs, un nœud compare les degrés de saturation reçus et envoie un message <color_yourself> à celui qui a le plus grand DS. Si le plus grand DS est détenu par plus d'un voisin, le nœud choisis celui qui a le plus grand degré entre eux. Si de plus plusieurs voisins ont le plus grand degré de saturation et ont le même degré, alors le nœud choisis celui qui a le plus grand ID (les ID sont unique, un seul est choisis forcément) et lui envoie un message <color_yourself> et le supprime de la liste des voisins actifs.

Lorsqu'un nœud reçoit un message <impass> et qu'il a une liste de voisins actifs non vide alors il envoie un message <ds_request>, aux voisins actifs. Dans le cas où il n'a pas de voisins actifs, il passe le message <impass> à son master.
Si après avoir envoyé des messages <ds_request> aux voisins actifs, un nœud reçoit un message <already_colored> alors le nœud ne fait qu'incrémenter un compteur pour considérer qu'il a reçu une réponse de sa requête.

Le pseudo-code que nous venons de décrire est donné dans la figure suivante :

Algorithme DSATUR_DISTRIBUE pour le nœud i

1 : $\boldsymbol{Const}: ID, D: int;$
$neighbors: set\ of\ neighbors;$
$\boldsymbol{Var}:\ DS: Saturation\ degree, \boldsymbol{init}\ 0\ ;$
$list\_color\ :\ set\ of\ available\ colors, \boldsymbol{init}\ \{C1, C2, \dots Cn\};$
$color: int, \boldsymbol{init}\ null;$
$neighbors\_actif: set\ of\ actif\ neighbors, \boldsymbol{init}:\ neighbors;$
$neighbors\_D, neighbors\_ID, neighbors\_DS\ remain, colored\_set, \boldsymbol{init}\ \emptyset;$
$START, colored, \boldsymbol{init}\ false;$
2 : $\boldsymbol{for}\ j\ in\ neighbors: send\ <my\_id>\ to\ j\ ;$
3 : $\boldsymbol{upon}\ receipt\ of\ <my\_id>\ from\ j:$
4 : $neighbors\_Id = \{neighbors\_Id, D(j)\};$
5 : $neighbors\_D = \{neighbors\_D, D(j)\};$
6 : $\boldsymbol{if} |neighbors\_id| = |neighbors| \boldsymbol{then}$
7 : $\boldsymbol{if} (D(i) > D(k)\ \forall\ k \in neighbors)$
$\vee\ (\exists k \in neighbors: D(i) = D(k) > D(k')\ \forall\ k' \in neighbors\ /\ k \wedge ID(i) > ID(k)) \boldsymbol{then}$
8 : $START = true;$
9 : $\boldsymbol{if}\ START = true\ \boldsymbol{then}$
10 : $color = list\_color[0];$
11 : $send < my\_color > to\ all\ neighbors$
12 : $\boldsymbol{if}\ \exists j \in neighbors: (D(j) > D(k) \forall k \in neighbors/j)$
$\vee\ \big(D(j) = D(k)\ \forall\ k \in neighbors/j \wedge ID(j) > ID(k)\big) \boldsymbol{then}$

13 : $remain = neighbors/j$;
14 : $actif\_neighbors = actif\_neighbors / j$;
15 : $colored\_set = \{colored\_set, j\}$;
16 : $send < color\_yourself > to\ j$;
$START = false$;
17 : **upon** $receipt\ of < color\_yourself > from\ neighbor\ j$:
18 : $master = j$;
19 : $color = list\_color[0]$;
20 : $remain = \{remain(j), remain\} / colored\_set$;
21 : $send < my\_color > to\ all\ neighbors$;
22 : **if** $|actif\_neighbors| \neq 0$ **then**
23 : $send < ds\_request > to\ j$;
24 : **else if** $|actif\_neighbors| = 0 \wedge |remain(j)| = 0$ **then**
25 : $send < impass > to\ j$;
26 : **upon** $receipt\ of < my\_color > from\ neighbor\ j$:
27 : $DS++$;
28 : $remain = remain/j$;
29 : $actif\_neighbors = actif\_neighbors / j$
30 : $colored\_set = \{colored\_set, j\}$;
31 : $list\_color = list\_color/color(j)$;
32 : **upon** $receipt\ of < ds\_request > from\ neighbor\ j$:
33 : $send < ds\_response > to\ j$;
34 : **upon** $receipt\ of < ds\_response > from\ neighbor\ j$:
35 : $neighbors\_DS = \{neighborDS\_DS(j)\}$;
36 : **if** $|neighbors\_DS| = |neighbors\_actif|$ **then**
37 : **if** $\exists k \in neighbors: (DS(k) > DS(k')\forall k \in neigbors/j), \forall k' \in neighbors/j, k)$
$\vee \big(DS(k) = DS(k')\ \forall\ k \in neighbors/j \wedge D(k) > D(k')\big)$
$\vee \big(DS(k) = DS(k') \wedge D(k) = D(k')ID(k) > ID(k')\big)$ **then**
38 : $remain = remain\ /\ k$;
39 : $actif\_neighbors = actif\_neighbors / k$;
40 : $colored\_set = \{colored\_set, k\}$;
41 : $send < color\_yourself > to\ j$;
42 : **upon** $receipt\ of < impass > fron\ neighbor\ j$:
43 : **if** $|neighbors\_actif| \neq 0$ **then**
44 : $remain = \{remain(j), remain\} / colored\_set$;
45 : $send < color\_yourself > to\ j$;
46 : **else** $send < impass > to\ master$;

L'algorithme que nous venons de décrire est un algorithme de coloriage de sommets alors que notre cas de figure demande une solution de coloriage d'arête, les nœuds ainsi colorié par DSATUR_DISTRIBUE ne seront pris en compte dans la réalité que pour des arêtes. Le passage d'un graphe représentant un réseau réel vers un graphe que nous appelons virtuel sera discuté dans la prochaine partie. Un exemple de l'exécution de l'algorithme DSATUR_DISTRIBUE proposé sera également donné dans la partie « Compte rendu du projet ».

5.2 Conception de l'algorithme distribué Edge_Coloring à vue locale

Cet algorithme nécessite comme l'algorithme précédant de considérer que les ID dans le réseau sont uniques. Egalement, aucun nœud n'a une vision globale du réseau. Dans cet algorithme de coloriage d'arêtes, chaque nœud dispose de la constante ID, de la variable de l'état indiquant s'il est initiateur **START**.

La première étape consiste à échanger des messages <**my_id**> entre un nœud et ses voisins, chaque nœud informe ses voisins de son ID ainsi que son degré et reçoit la liste des ID ainsi que les degrés de ses voisins. A l'issue de cette étape, chaque nœud sait s'il a le plus grand ID par rapport à ses voisins.

Après avoir reçu les identifiants de tous ses voisins, un nœud considère qu'il est initiateur s'il a le plus grand ID (plus d'un nœud peuvent être initiateurs) et met la variable START = true et envoie des messages <Propose> à tous ses voisins contenant la couleur proposée tel que la plus petite couleur est proposée au voisin ayant le plus grand ID parmi l'ensemble des voisins.

A la réception du message <Propose>, un nœud vérifie si la couleur proposée existe dans la liste des couleurs disponible, si c'est le cas alors il considère que le lien est colorié avec la couleur en question, supprime cette couleur immédiatement de la liste des couleurs disponibles, répond par un message <Accept> et cherche à colorier à son tour les liens avec ses voisins en envoyant des message <Propose> de la même manière que pour l'initiateur, sinon si la couleur proposée n'est pas disponible, alors il envoie un message <Refuse> contenant la couleur refusée.

Après avoir reçu le message <Accept>, le nœud considère que le lien est colorié avec la couleur en question et supprime cette couleur immédiatement de la liste des couleurs disponibles.

Si un nœud reçoit un message <Refuse>, alors il propose une nouvelle couleur à son voisin tel que la couleur qui sera reproposée sera égale au prochain élément disponible dans la liste list_color (prochain élément modulo taille de la liste des couleurs).
Après avoir colorié les liens avec tous ses voisins, le nœud peut arrêter l'exécution de l'algorithme.

Le pseudo-code que nous venons de décrire est donné dans la figure suivante :

Algorithme Edge_Coloring pour le nœud i

1 : ***Const*** : $ID : int$;
$neighbors$: $set\ of\ neighbors$;
Var : $list\_color$: $set\ of\ available\ colors$, ***init***$\{C1, C2, \dots Cn\}$;
$neighbors\_id$, ***init*** $\emptyset$;
$higher$: $number\ of\ neighbors\ with\ higher\ id$, ***init*** 0;
$num\_accepted$: $number\ of\ colored\ links$, ***init*** 0;

```
                neighbor_color: list of colors proposed to neighbors, init ∅
                START: boolean, init false;
2   :   for j in neighbors : send < my_id > to j ;
3   :   upon receipt of < my_id > from neighborj :
4   :       neighbors_Id = {neighbors_Id, D(j)};
5   :       if ID(i) < ID(j) then
6   :           higher + +;
7   :       if |neighbors_id| = |neighbors| ∧ ID(i) > ID(k) ∀k ∈ neighbors then
8   :           START = true;
9   :   if START = true then
10  :       for j ∈ neighbors:
11  :           neighbor_color[j] = list_color[(|list_color| − j − 1)%|list_color|];
12  :           send < propose(neighbor_color[j]) > to j;
13  :           START = false;
14  :   upon receipt of < propose(proposed_color) > from neighbor j:
15  :       if proposed_color ∈ list_color then
16  :           color_link(i, j) = proposed_color;
17  :           list_color = list_color/proposed_color ;
18  :           send < accept(accepted_color) > to j;
19  :           num_accept + +;
20  :       else send < refuse(refused_color) > to j;
21  :       if num_accept = higher then
22  :           for k ∈ neighbors: ID(i) > ID(k):
23  :               neighbor_color[k] = list_color[(|list_color| − k − 1)%|list_color|];
24  :               send < propose(neighbor_color[k]) > to j;
25  :   upon receipt of < accept(accepted_color) > from neighbor j:
26  :       color_link(i, j) = accepted_color;
27  :       list_color = list_color/accepted_color ;
28  :   upon receipt of < refuse(refused_color) > from neighbor j:
29  :       neighbor_color[k] = list_color[(|list_color| − k)%|list_color|];
30  :       send < propose(neighbor_color[k]) > to j;
```

Un exemple de l'exécution de l'algorithme DSATUR_DISTRIBUE proposé sera également donné dans la partie « Compte rendu du projet ».

5.3 Conception de l'algorithme de coloriage aléatoire d'arêtes

Cet algorithme est simple à réaliser et va nous servir comme référence pour comparer les performances des algorithmes que nous proposons. Il s'exécute en deux étapes seulement. Du fait que les nœuds ne connaissent pas leurs voisins, alors chaque nœud envoie un message <my_id> à son voisin. Ensuite, un nœud qui a un plus grand ID que son voisin choisi une couleur aléatoirement pour le lien qui les relient et lui envoie un message <color> qui contient la couleur prise. En recevant le message <color>, le nœud considère à son tour la couleur donnée pour colorier le lien. Il est évident que cet algorithme ne garantit pas d'éviter d'attribuer la même couleur pour deux arêtes adjacentes.

6 Compte rendu du projet

Dans cette dernière partie du rapport, nous allons d'abord dérouler chacun des trois algorithmes dans une topologie donnée afin de bien illustrer leur comportement. Ensuite, nous allons montrer l'approche prise pour adapter le DSATUR pour un coloriage distribué d'arêtes. Enfin, les courbes de performances (nombre de conflit, temps de convergence et nombre de message) en fonction du nombre de nœuds pour 3 cas (réseau moyennement dense, réseau dense et réseau très dense) seront données avec une comparaison. Nous clôtureront cette partie par une conclusion.

6.1 Exemple pour l'algorithme DSATUR_DISTRIBUE

Supposons le graphe à colorier sur la figure ci-dessous, la première étape consiste à échanger des messages <my_id> contenant l'ID et le degré entre chaque nœud et ses voisins.

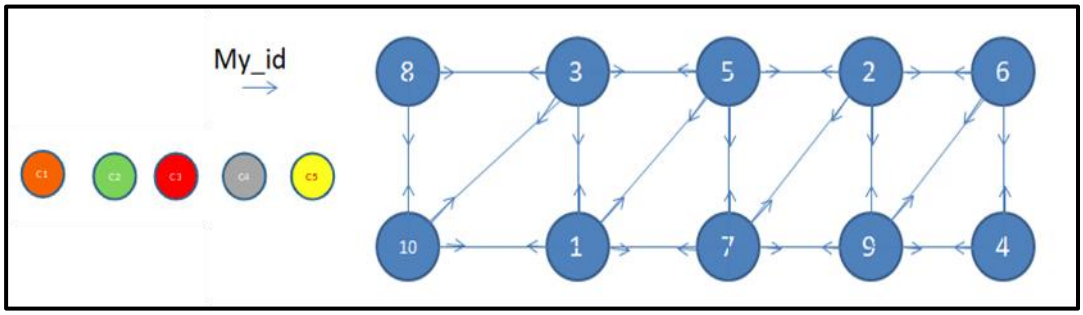

Figure 6.1 Echange de messages my_id.

Les nœuds 8, 10, 6 et 4 n'ont pas le plus grand ID par rapport à leurs voisins respectifs, ils ne sont donc pas initiateurs. Le nœud 3 a un degré maximal dans son voisinage (degré 4) mais un de ses voisins, le nœud 5, possède également le même degré et a un ID plus grand, par conséquent le nœud 3 ne peut être un initiateur. Les nœuds 1, 5, 7 et 2 sont dans la même situation que le nœud 3. Le nœud 9 quant à lui possède le plus grand degré dans son voisinage, et bien que deux de ses voisins (nœuds 3 et 7) ont le même degré mais possèdent un ID inférieur à 9, donc le nœud 9 décide qu'il est initiateur.

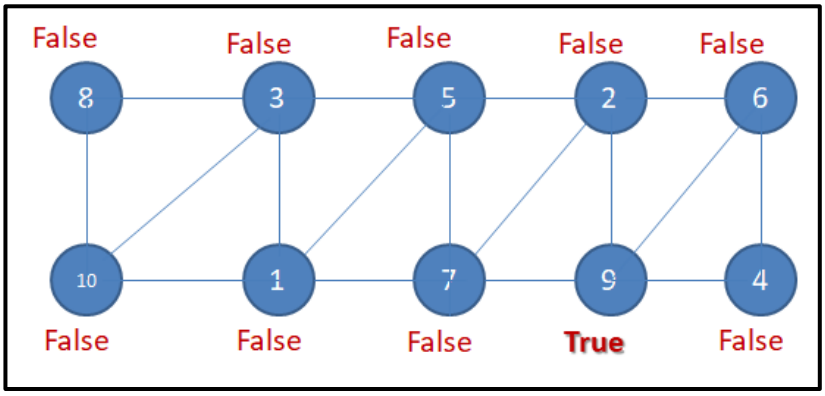

Figure 6.2 Décision du nœud initiateur.

Le nœud 9 prend la plus petite couleur disponible et informe ses voisins (les nœuds 7, 2 6 et 4 qu'ils ne peuvent désormais pas utiliser la couleur C1). Du fait qu'il connait déjà les degrés de ses voisins et que leurs degrés de saturation sont nuls (le coloriage n'a pas encore commencé), alors il envoie un message color_yourself au voisin qui a le plus grand degré (le nœud 7 ou 2), comme le nœud 7 est plus grand que 2 alors il sera choisi par le nœud 9 qui lui envoie un message color_yourself précisant dans la liste remain qu'il reste les nœuds 2, 6 et 4 à colorier.

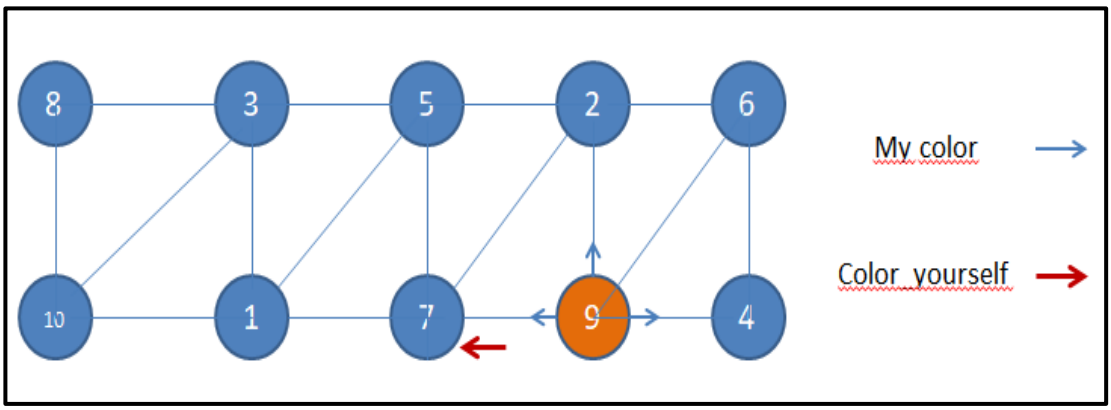

Figure 6.3 Envoie du message color_yourself.

Le nœud 7 prend la plus petite couleur disponible C2 et envoie le message my_color à tous ses voisins actifs (nœuds 1, 5 et 7) ainsi que le message ds_request. Chacun de ces voisins (5, 1 et 2) éliminent la couleur C2 de la liste des couleurs disponibles et le nœud 7 de la liste des voisins actifs et envoient une réponse à la requête ds_request précisant leurs degrés de saturation. Ainsi, les nœuds 1 et 5 ont un DS=1 car ils ont reçu un seul message my_color (celui du nœud 7), tandis que le nœud 2 possède un DS égal à 2 (il a 2 voisins coloriés : 9 et 7). Par conséquent, le nœud 7 envoie un message color_yourself au nœud 2 avec la liste remain contenant les nœuds 4 ,6 ,1 et 5. Ces étapes sont illustrées dans la figure ci-dessous :

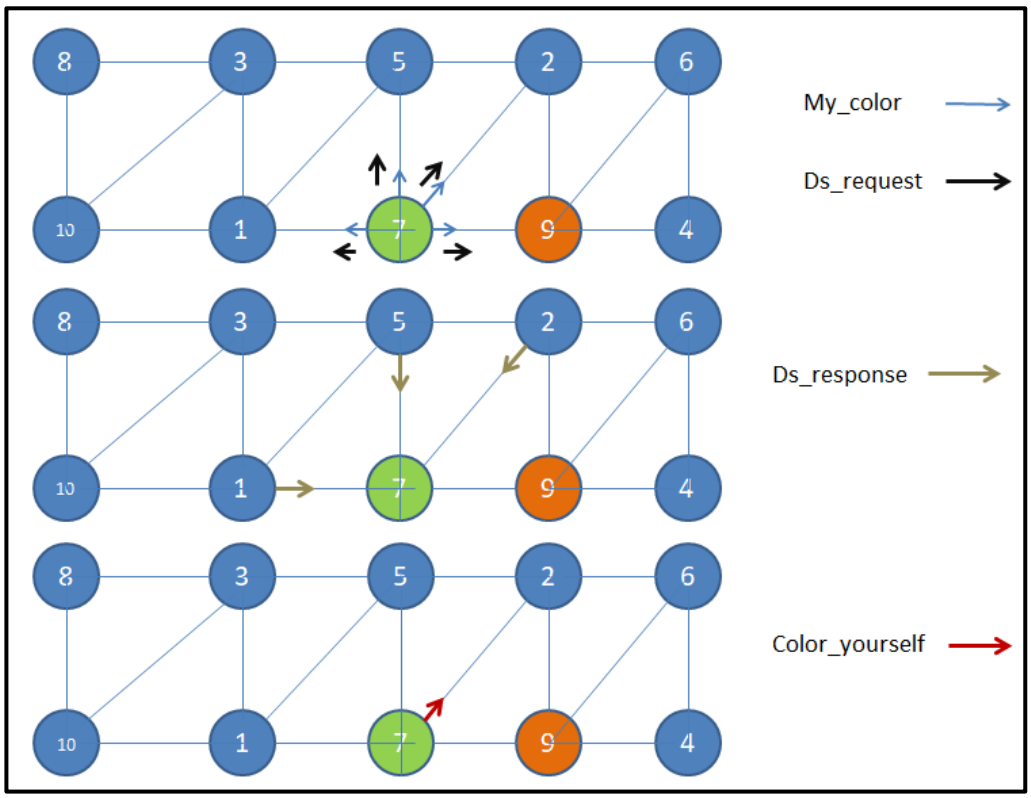

Figure 6.4 Mécanisme DS_request et DS_response dans DSATUR_DISTRIBUE.

A la réception du message color_yourself, le nœud 2 prend la plus petite couleur disponible C3, informe ses voisins et envoie un message ds_request aux nœuds 5 et 6, les deux ayant le même degré de saturation égal à 2, le nœud 2 choisis celui qui a le plus grand degré (le nœud 5) et lui envoie le message color_yourself contenant dans la liste remain les nœuds 1, 4 et 6.
De la même manière, à la réception du message color yourself, chaque nœud met à jour la liste des nœuds restant remain et la liste des couleurs coloried_set, demande les degrés de ses voisins actifs et choisis l'un d'entre eux. Voici un résumé des étapes de l'exécution :

- Nœud 2 prend la couleur C3 et envoie au nœud 5 color_yourself avec remain = {1, 6, 4}
- Nœud 5 prend la couleur C1 et envoie au nœud 1 color_yourself avec remain = {3, 6, 4}
- Nœud 1 prend la couleur C3 et envoie au nœud 3 color_yourself avec remain = {10, 8, 6, 4}
- Nœud 3 prend la couleur C2 et envoie au nœud 10 color_yourself avec remain = {8, 6, 4}
- Nœud 10prend la couleur C1 et envoie au nœud 8 color_yourself avec remain = {6, 4}

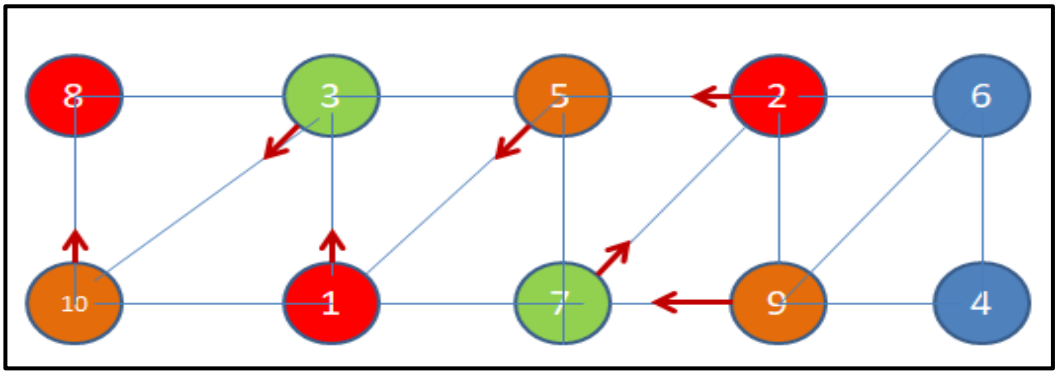

Figure 6.5 Déroulement de l'algorithme DSATURE_DISTRIBUE

Lorsque le nœud 8 reçoit le message color_yourself, il constate qu'il n'a plus de voisins actifs mais la liste remain n'est pas vide, il envoie alors à son master un message impass (contenant également la liste ramain). En recevant le message impass, chacun des nœuds 10, 3, 1, et 5 n'ont plus de voisin actifs, chacun d'entre eux repasse le message impass à son master. Lorsque le message impass arrive au nœud 2, celui-ci possède un voisin actif, il ne repasse donc pas le message impass à son master mais envoie un message color_yourself (après ds_request et ds_response) au nœud 6. Le nœud 6 prend donc la couleur C2 et ordonne au nœud 4 de prendre un couleur. Enfin, le nœud 4 prend la couleur C3 et constate qu'il ne possède pas de voisins actifs et que la liste remain est vide, l'algorithme arrive donc à sa fin.

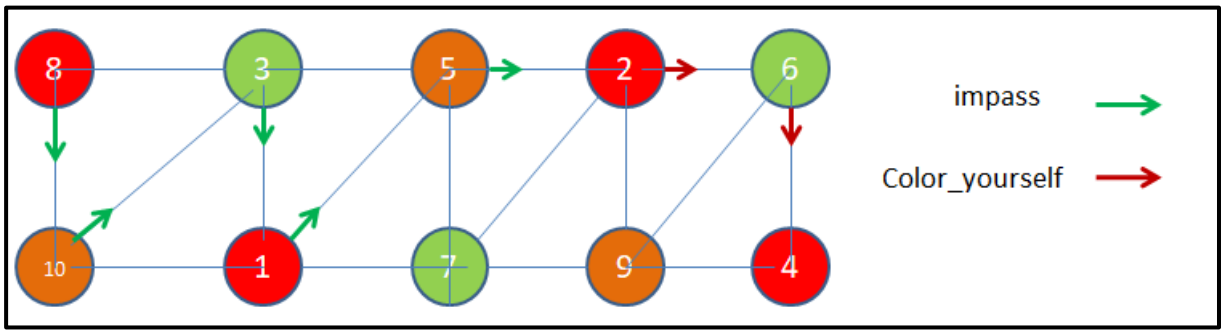

Figure 6.6 Mécanisme Impass dans DSATUR_DISTRIBUE.

Lorsqu'un nœud non colorié n'a plus de couleur disponible, alors un évènement de conflit se présente.
La figure ci-dessous montre l'exécution de l'algorithme DSATUR_DISTRIBUE sur le simulateur OMNET++ avec un graphe aléatoire de 20 nœuds et de connectivité de 50%.

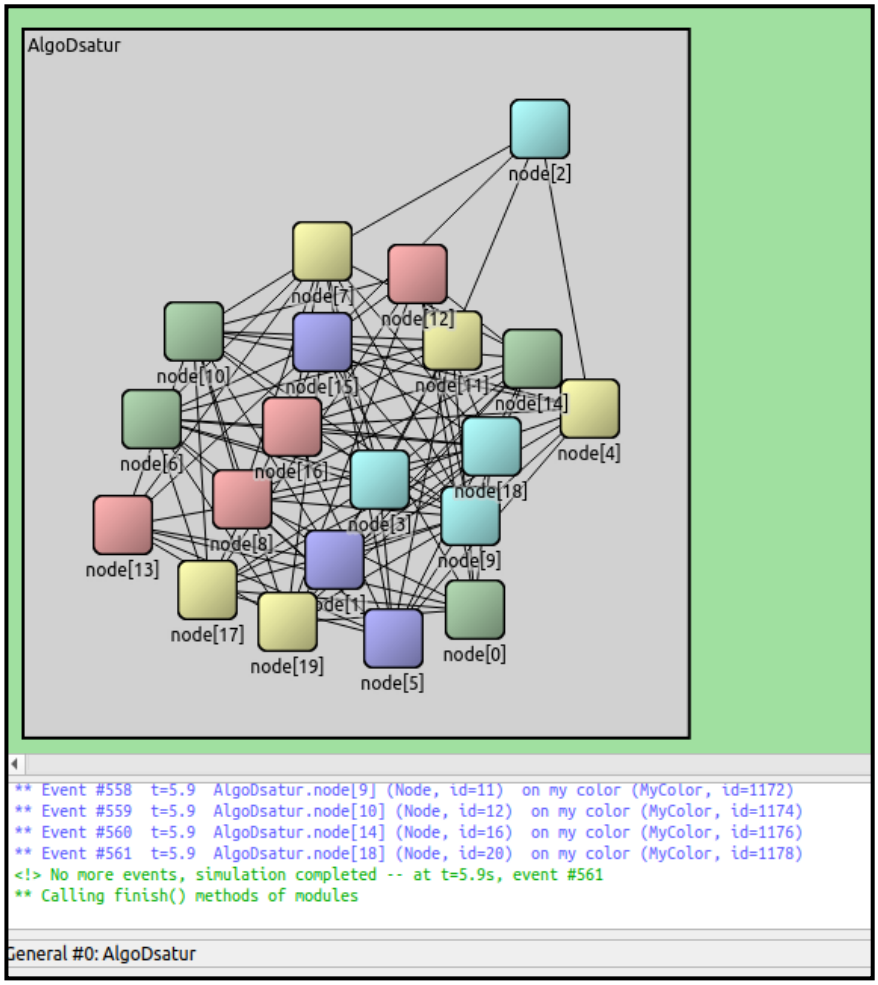

Figure 6.7 Exécution de l'algorithme DSATUR_DISTRIBUE sur OMNET++.

6.2 Adaptation de l'algorithme DSATUR_DISTRIBUE pour un coloriage d'arêtes

Le problème traité dans ce projet concerne un coloriage d'arêtes, cependant l'algorithme DSATUR_DISTRIBUE réalise un coloriage de nœuds. En réalité, le graphe sur lequel s'applique l'algorithme n'est qu'un graphe où les nœuds représentent les arêtes du graphe réel. Afin de bien comprendre l'idée, nous présentons l'exemple du graphe suivant :

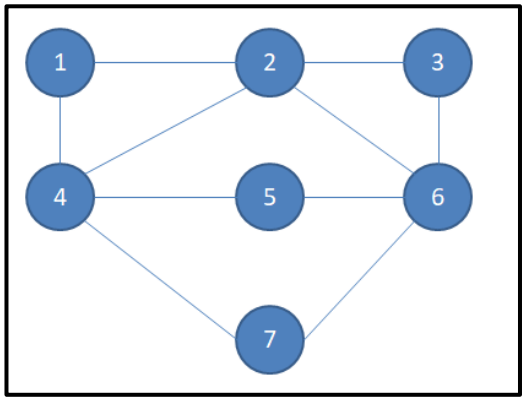

Figure 6.8 Graphe réel à transformer.

Dans ce graphe réel, chaque nœud doit connaitre la liste de ses voisins, et également les voisins de tous ses voisins. Deux étapes sont donc nécessaires pour obtenir ces informations, la première où chaque nœud envoie son ID à son voisin, et la deuxième, une fois que chaque nœud possède la liste de ses voisins il la transmet vers tous ses voisins. Ainsi, le nœud 1 a une connaissance des liens suivant {(1,2), (1,4), (2,3), (2,4), (2,6), (4,5), (4,7)}. Il est à noter que le lien est nommé de la manière suivante : Le plus grand identifiant à droite et le petit identifiant à gauche, ce détail est important pour décider quel nœud se charge d'un lien donné. En effet, un nœud prend en charge de colorier les liens dont la partie gauche de l'identifiant correspond à l'identifiant du nœud (ie. Le nœud 1 prend en charge de colorier les liens (1,2) et (1,4)) tout en prenant en considération un sous graphe où les nœuds sont représentés par la liste des liens dont il a connaissance (liste donnée en dessus), et les liens sont établit dans ce graphe lorsque une partie des identifiants du lien (gauche ou droite) est commune entre deux éléments de la liste.
De manière plus claire, le nœud 1 a une vision sur le graphe suivant :

- Liste des nœuds (virtuels) = V = {(1,2), (1,4), (2,3), (2,4), (2,6), (4,5), (4,7)}
- Liste des liens (virtuels) = E = {[(1,2), (1,4)], [(1,2), (2,3)], [(1,2), (2,4)], [(1,2), (2,6)], [(1,4), (2,4)], [(1,4), (4,7)], [(1,4), (4,5)]}
- Nœuds virtuels que le nœud 1 prend en charge : (1,2) et (1,4)

Pratiquement, à l'exécution de l'algorithme DSATUR_DISTRIBUE, à chaque changement dans un nœud virtuel, un message est envoyé au nœud réel voisin qui a connaissance du nœud virtuel en question.

De la même manière, chaque nœud construit un sous graphe virtuel localement. Nous donnons ci-dessous une figure qui résume la démarche à suivre :

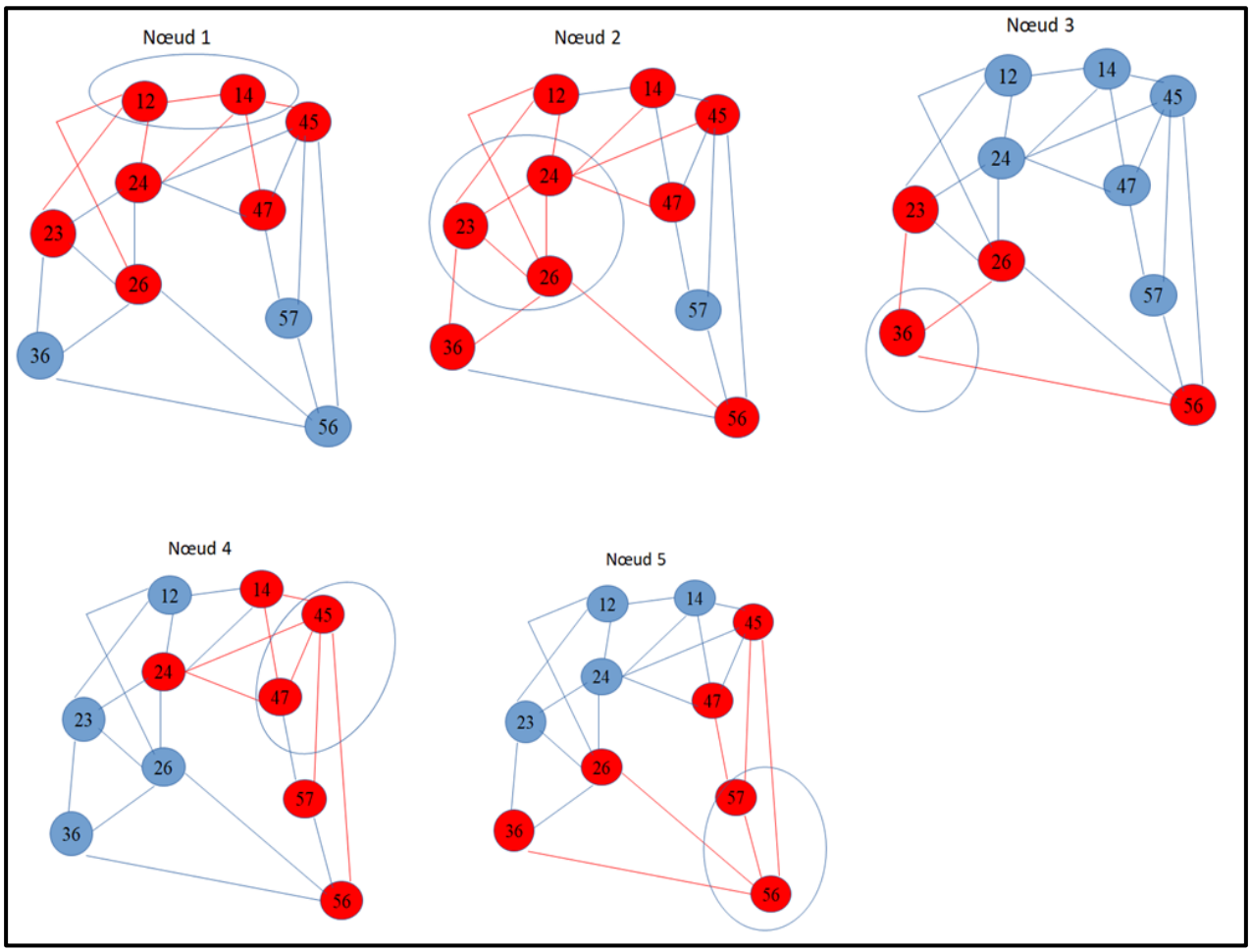

Figure 6.9 Sous graphes virtuels vus par les nœuds du graphe réel.

6.3 Exemple pour l'algorithme distribué Edge_Coloring à vue locale

Comme pour l'exemple précédant, supposons le graphe à colorier sur la figure ci-dessous, la première étape consiste à échanger des messages <my_id> contenant l'ID et le degré entre chaque nœud et ses voisins.

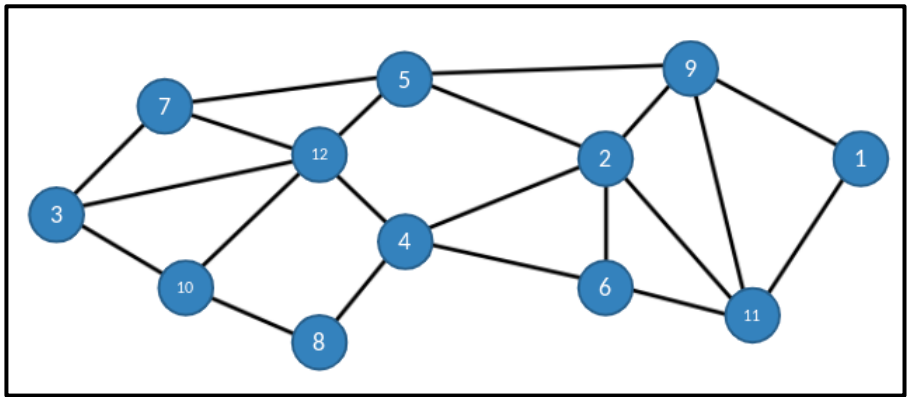

Figure 6.10 Graphe initial à colorier.

Après cette première étape d'échanges d'identifiant entre voisins directs, seuls les nœuds 12 et 11 décident qu'ils sont initiateurs car ils ont le plus grand degré dans leurs voisinages respectifs. Chacun d'eux propose donc une couleur pour colorier les liens avec chacun de ses voisins. La plus petite couleur est proposée au voisin ayant le plus grand ID. Le nœud 12 propose C1 au nœud 10, C2 au nœud 7, C3 au nœud 5, C4 au nœud 4 et C5 au nœud 3. De la même manière, le nœud 11 propose C1 au nœud 9, C2 au nœud 6, C3 au nœud 2 et C4 au nœud 1.

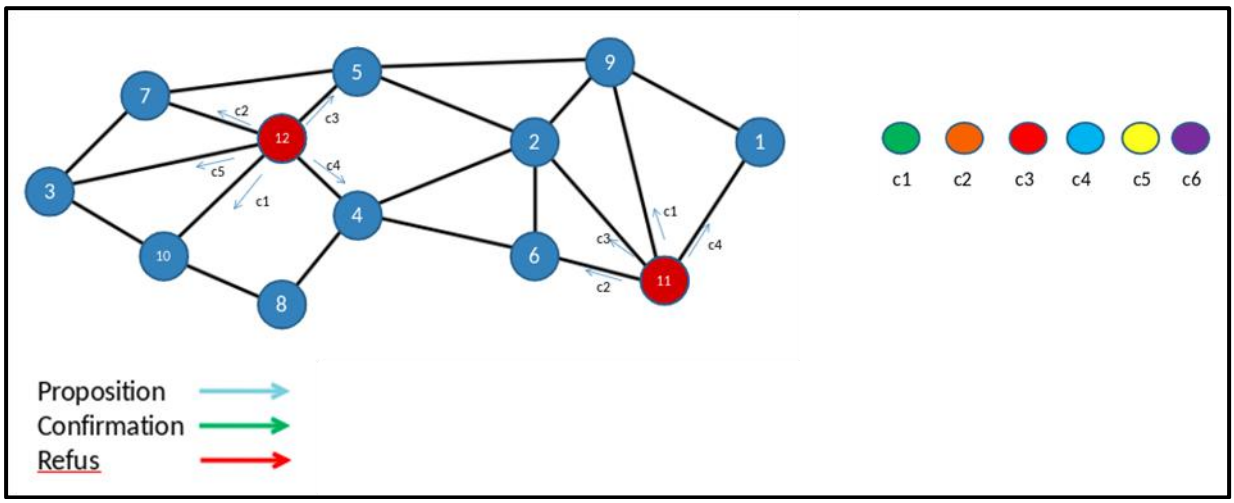

Figure 6.11 Démarrage de l'algorithme Edge_Coloring par les nœuds initiateurs.

A la réception de la proposition, chaque nœud accepte la couleur proposée si elle fait partie des couleurs disponibles en considérant le lien colorié par cette couleur, supprime la couleur de la liste des couleurs disponibles et envoie un message accepte.

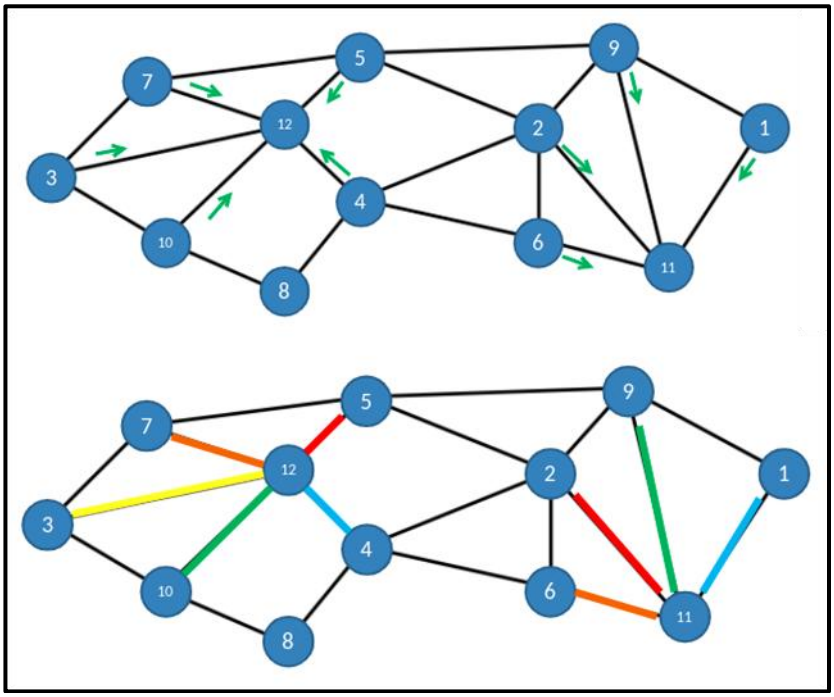

Figure 6.12 Etablissement de la couleur du lien par Edge_Coloring.

Lorsque les nœuds 12 et 11 reçoivent un message Accept, ils considèrent le lien colorié aves la couleur proposée.
Après avoir colorié tous les liens avec des voisins ayant un plus grand ID, un nœud passe ensuite au coloriage des liens avec les nœuds qui ont un ID plus petit que le sien.
Les nœuds 7, 10, 6 et 9 ont donc le plus grands ID dans leurs voisinages (voisins_actifs).
Le nœud 7 ne peut pas proposer C2 car elle n'est pas disponible (elle est déjà utilisée), il propose C1 au nœud 5 et C3 au nœud 3. De la même manière, le nœud 10 propose C2 au nœud 8 et C3 au nœud 3. Le nœud 9 propose C2 au nœud 5, C3 au nœud 2 et C4 au nœud 2. Le nœud 6 propose C1 au nœud 4 et C2 au nœud 2.

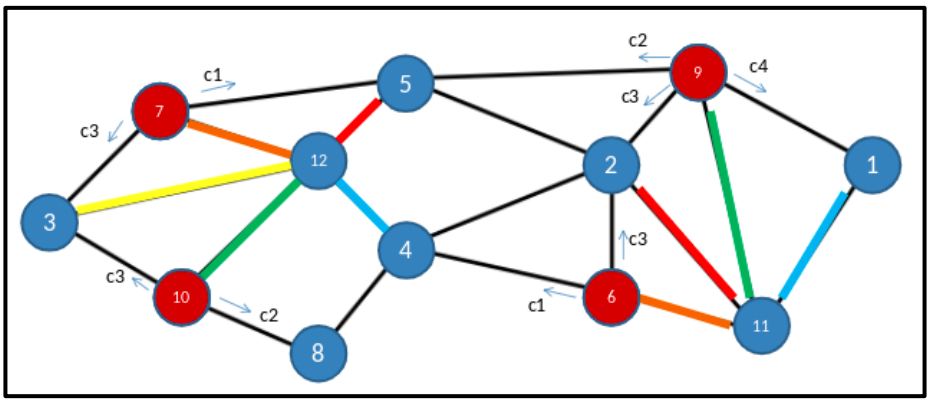

Figure 6.13 Reprise du processus de coloriage dans l'algorithme Edge_Coloring.

Le nœud 3 reçoit simultanément deux propositions de la même couleur C3, il accepte donc l'une d'entre elles (celle reçue en premier), et refuse l'autre. D'autre part, le nœud 2 refuse les deux propositions car C3 n'est pas disponible, le nœud 1 refuse C4 et les nœuds 4 et 8 acceptent les propositions.

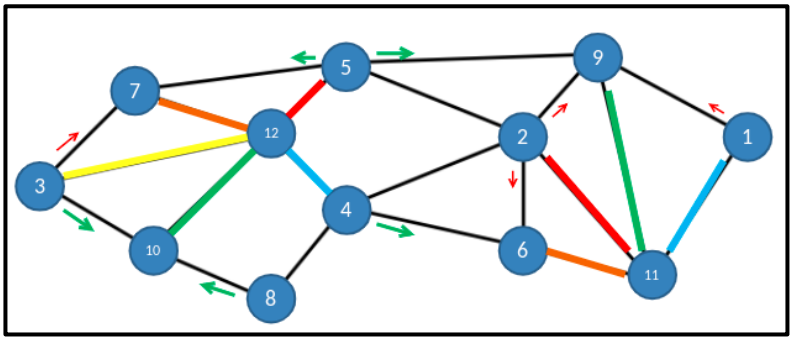

Figure 6.14 Refus et acceptation des propositions dans Edge_Coloring.

Lorsque le nœud 7 reçoit le refus de la couleur C3 proposée au nœud 3, il propose donc la prochaine couleur disponible C4. Les nœuds 6 et 9 proposent C4, C4 et C3 aux nœuds 2 et 6 respectivement.
Les mêmes étapes sont exécutées pour le reste des nœuds jusqu'à ce que tous les nœuds voient tous leurs liens incidents coloriés. La figure ci-dessous résume le restant des étapes :

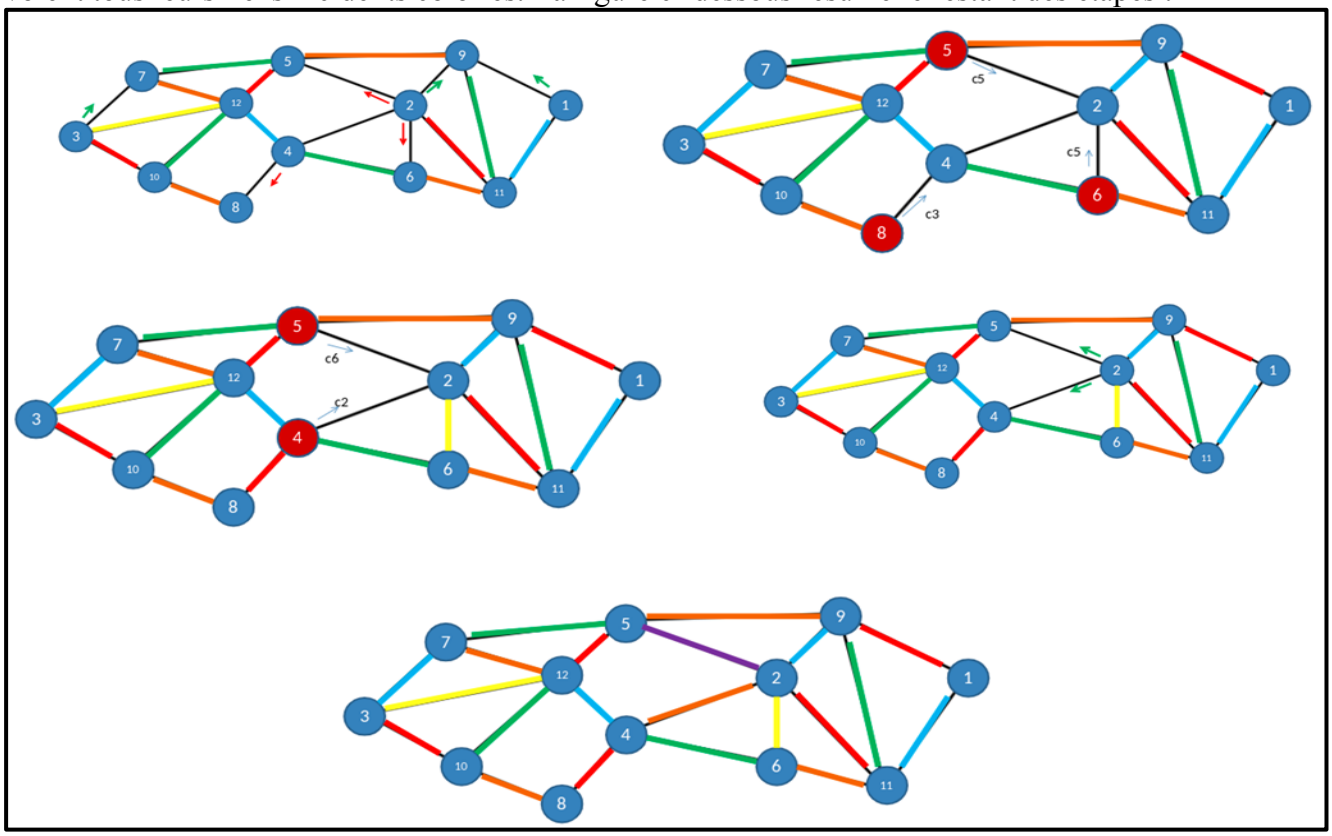

Figure 6.15 Déroulement de l'algorithme Edge_Coloring.

Lorsqu'un nœud n'a plus de couleur disponible à proposer à un de ses voisins, alors un évènement de conflit se présente.
La figure ci-dessous montre l'exécution de l'algorithme Edge_Coloring sur le simulateur OMNET++ avec un graphe aléatoire de 20 nœuds et de connectivité de 20%.

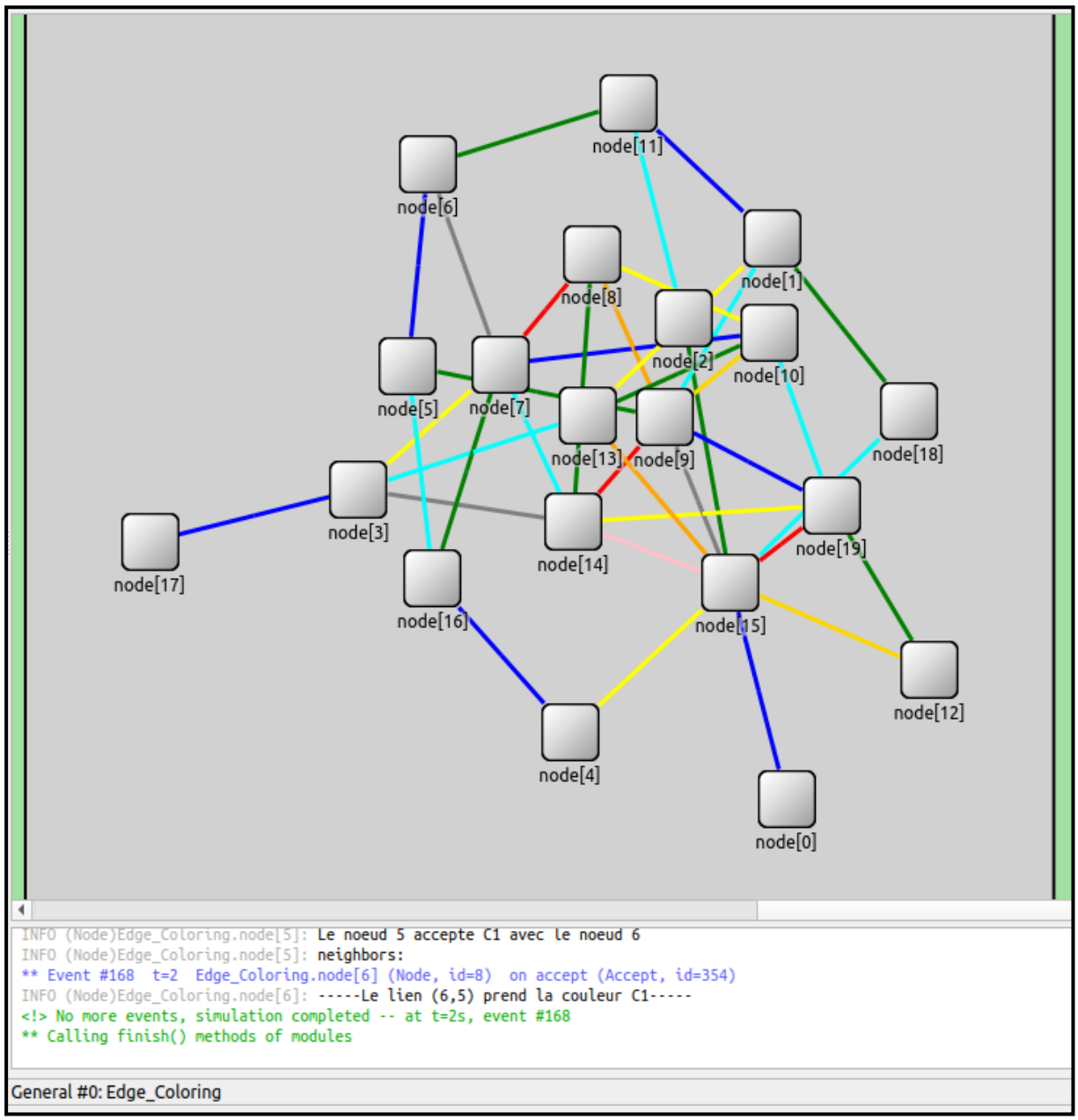

Figure 6.16 Exécution de l'algorithme Edge_Coloring sur OMNET++.

6.4 Exemple pour l'algorithme de coloriage aléatoire des arêtes Random_Coloring

Cet algorithme est simple, néanmoins une phase d'échange d'ID entre les voisins directs est nécessaire. Une fois l'ID du voisins reçu, un nœud choisis une couleur aléatoirement disponible pour les deux extrémités pour colorier le lien et envoie le message <color> pour informer le voisin de la couleur qui a été considérée.
A la réception du message, un nœud n'a qu'à considérer la couleur donnée pour colorier le lien. L'algorithme s'exécute donc en deux étapes : échange d'ID et message de coloriage. La figure ci-dessous montre un exemple.

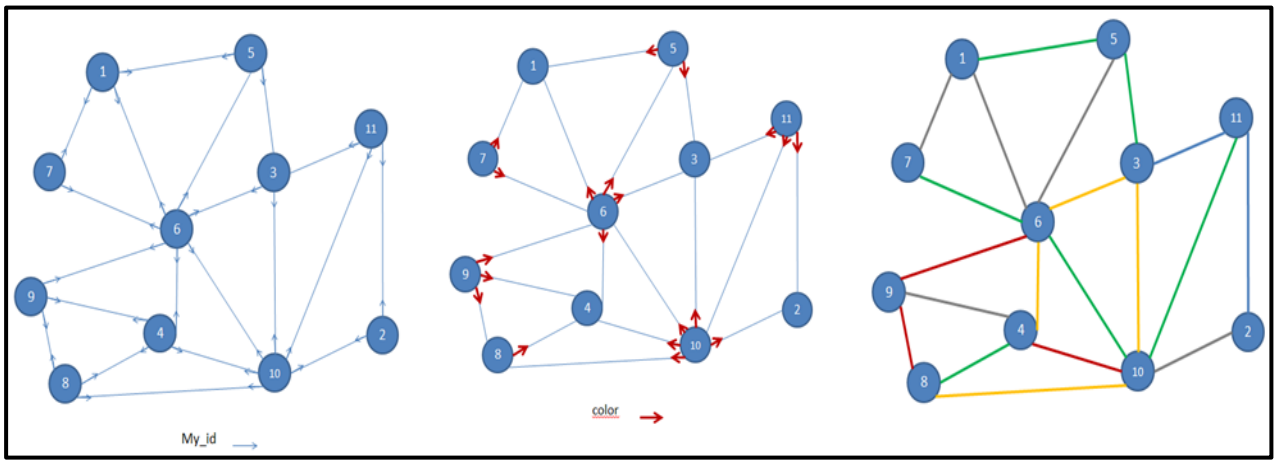

Figure 6.17 Exemple d'exécution de l'algorithme Random_Coloring.

Un évènement de conflit se présente lorsque deux arêtes adjacentes prennent la même couleur. Cet algorithme ne possède aucun mécanisme pour éviter d'atribuer deux couleurs à deux voisins adjacents.
La figure ci-dessous montre l'exécution de l'algorithme Random_Coloring sur le simulateur OMNET++ avec un graphe aléatoire de 20 nœuds et une connectivité de 50%.

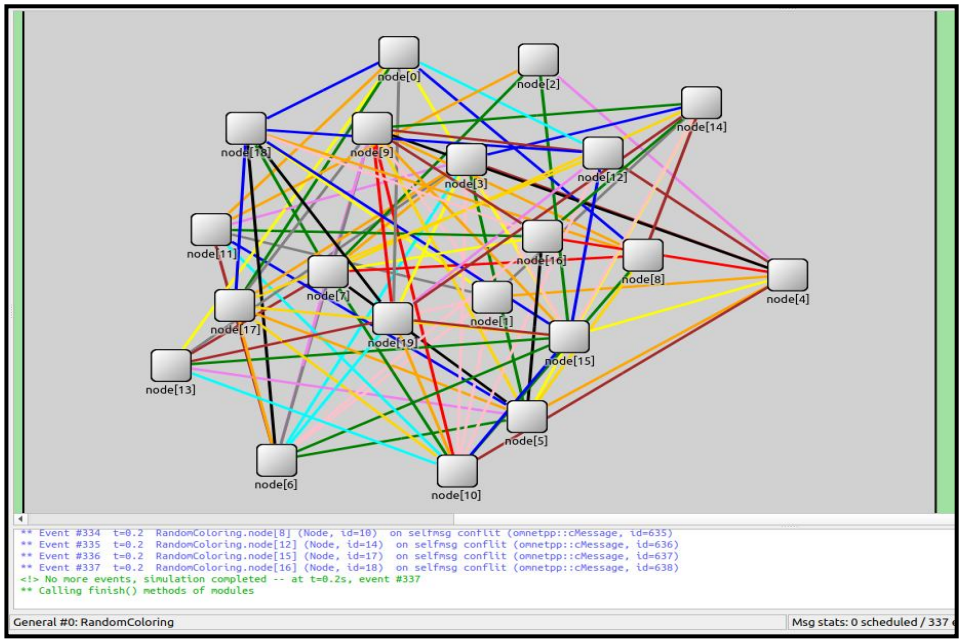

Figure 6.18 Exécution de l'algorithme Random_Coloring sur OMNET++.

6.5 Etude des performances des 3 algorithmes

Dans cette partie, nous allons discuter les performances en termes de nombre de conflits, de temps de convergence et de nombre de messages de chacun des trois algorithmes précédents. Ces trois critères sont étudiés suivant le nombre de nœuds du réseau et dans 3 situations différentes : réseau moyennement dense, réseau dense et réseau très dense. En effet, la connectivité du réseau est un facteur aussi important que le nombre de nœuds pour l'étude des performances. Nous avons donc pris un nombre de nœuds compris entre 5 et 100 avec un pas de 5 nœuds et pour chaque valeur du nombre de nœuds, les trois cas de densité du réseau sont considérés. Le nombre de couleurs disponibles est fixé à 12 couleurs pour toutes les simulations.

Les performances sont relevées grâce au simulateur OMNET++ et la topologie du réseau est programmée par le langage de description du réseau (NED) propre à l'outil [8]. Cette topologie est créée de manière aléatoire en spécifiant comme paramètre le nombre de nœuds et la probabilité de création d'un lien. Le retard de propagation dans le canal a été fixé d'une valeur de 100ms.

Pour définir le caractère de densité du réseau, il est à noter que la connectivité d'un réseau est un facteur qui doit être adapté selon le nombre de nœuds. En effet, une connectivité de 50% pour un nombre total de nœuds petit représente une connectivité moyenne, tandis que pour un grand nombre de nœuds, cette connectivité signifie un réseau très dense. Pour cette raison, nous avons adapté la valeur de la connectivité selon les trois cas décrits dans le tableau suivant.

Nombre de nœuds	Réseau moyennement dense	Réseau dense	Réseau très dense
<10	0.5	1	/
Entre 11 et 20	0.3	0.6	0.9
Entre 21 et 30	0.25	0.5	0.75
Entre 31 et 40	0.2	0.4	0.6
Entre 41 et 50	0.15	0.3	0.45
Entre 51 et 60	0.12	0.2	0.24
Entre 61 et 70	0.08	0.14	0.18
>71	0.06	0.1	0.15

Tableau 6.1 Définition du caractère de densité d'un réseau.

Comme expliqué dans les sections précédentes, l'algorithme DSATUR_DISTRIBUE établit un coloriage de nœud et est adapté pour un coloriage d'arêtes. Afin de pouvoir le comparer avec les deux autres algorithmes, il est impératif de considérer les mêmes propriétés du graphe (i.e. connectivité et nombre de nœuds). Il faut donc trouver une relation entre un graphe réel et un graphe virtuel qui lui correspond (où les liens sont représentés par des nœuds).

Considérons les notations suivantes :
n1 = Le nombre de nœuds du graphe réel.
m1 = Le nombre de liens du graphe réel.
p1 = La connectivité du graphe réel. (Entre 0 et 1)
n1 = Le nombre de nœuds du graphe virtuel.
m1 = Le nombre de liens du graphe virtuel.
p1 = La connectivité du graphe virtuel. (Entre 0 et 1)

Les paramètres pris par le programme pour la construction de la topologie sont n1, p1, n2 et p2. Il faut donc trouver les valeurs de n2 et p2 qui correspondent à n1 et p1.
Nous avons comme données n1, p1 et m1.

- Les nœuds du graphe réel sont représentés pas des liens, donc : n2 = m1
- Nous avons la relation suivante qui donne le nombre de liens du graphe virtuel :

$$m2 = \frac{1}{2}\sum_{v1 \in V1} \deg(v1) - m1$$

où v1 est un nœuds dans le graphe réel.

Le degré de chaque nœud du graphe réel n'étant pas connu, nous prenons donc une valeur moyenne pour tous les degrés en fonction de la connectivité du réseau selon la relation suivante :

$$moy(\deg(v1)) = p1\,.\,(n1 - 1)$$

$$m2 = \frac{1}{2}\sum_{v1 \in V1} (moy(\deg(v1)))^2 - m1 \quad = \frac{1}{2}\sum_{v1 \in V1} \left(p1.(n1-1)\right)^2 - m1$$

$$m2 \quad = \frac{1}{2}.n1.\left(p1.(n1-1)\right)^2 - m1$$

- D'autre part, le nombre de liens dans le graphe virtuels est donné par la relation suivante :

$$m2 = p2\,.\ \frac{n2.(n2-1)}{2}$$

- On peut obtenir donc la relation de p2 en fonction de n1 et m1 de la manière suivante :

$$\frac{1}{2}.n1.\left(p1.(n1-1)\right)^2 - m1 = \ p2\,.\ \frac{n2.(n2-1)}{2}$$

$$p2 = \frac{n1.(p1(n1-1))^2}{n2.(n2-1)} = \frac{n1.(p1(n1-1))^2}{m1.(m1-1)}$$

6.5.1 Performances pour un réseau moyen

Nous présentons dans ce qui suit les courbes du nombre de conflits, du nombre de message et du temps en fonction du nombre de nœuds dans un réseau moyennement dense.

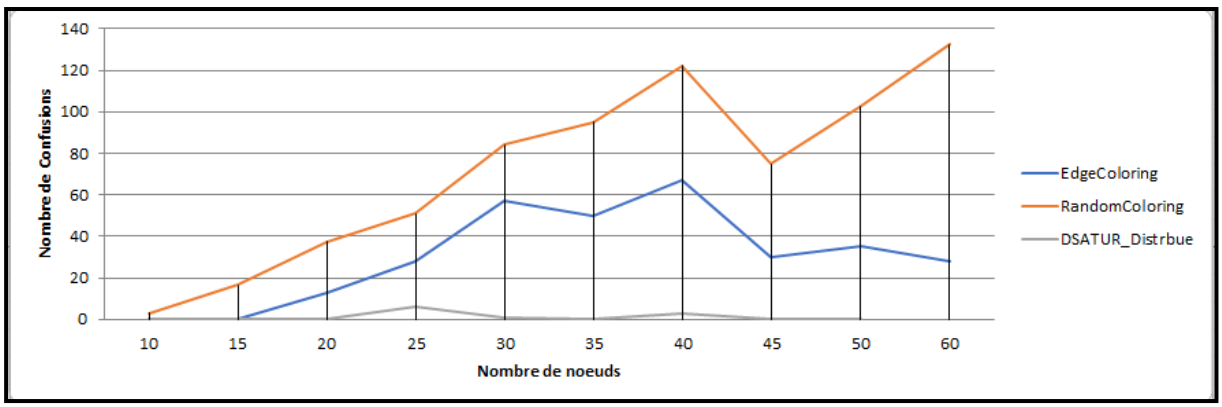

Figure 6.19 Nombre de conflit dans un réseau moyen.

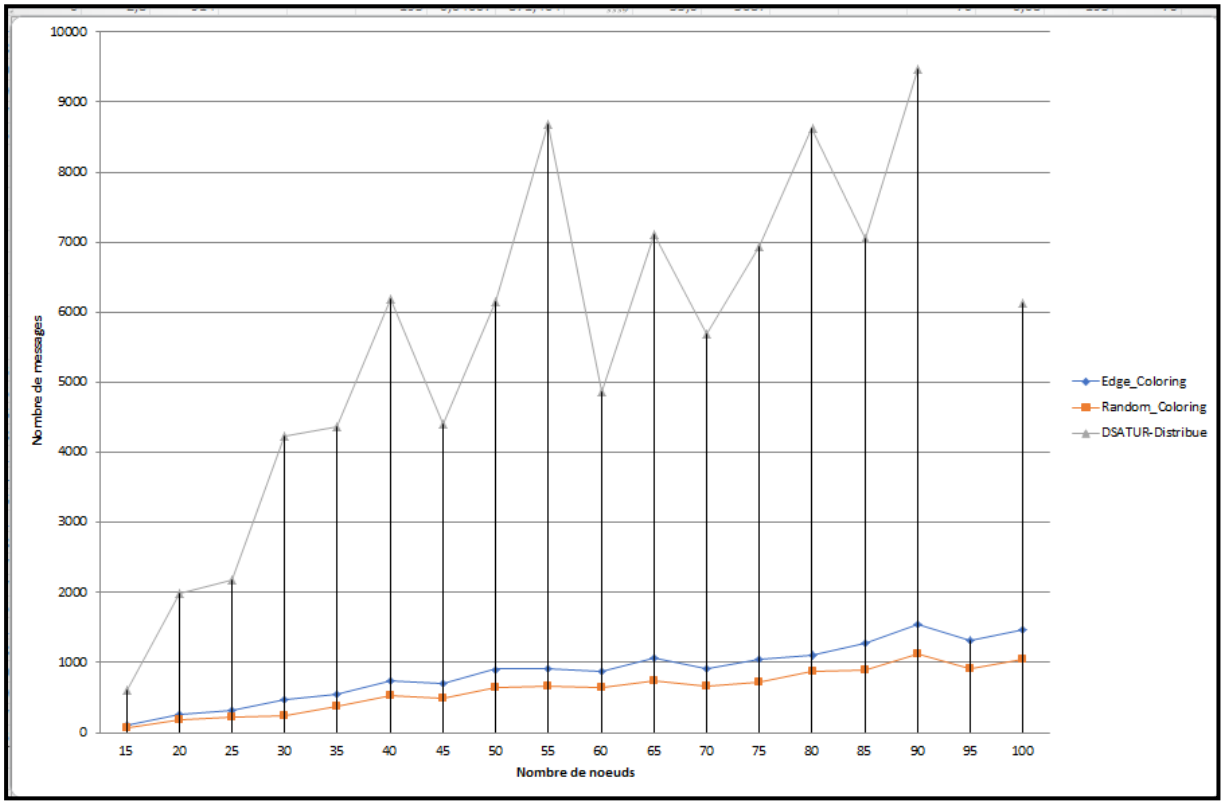

Figure 6.20 Nombre de messages dans un réseau moyen.

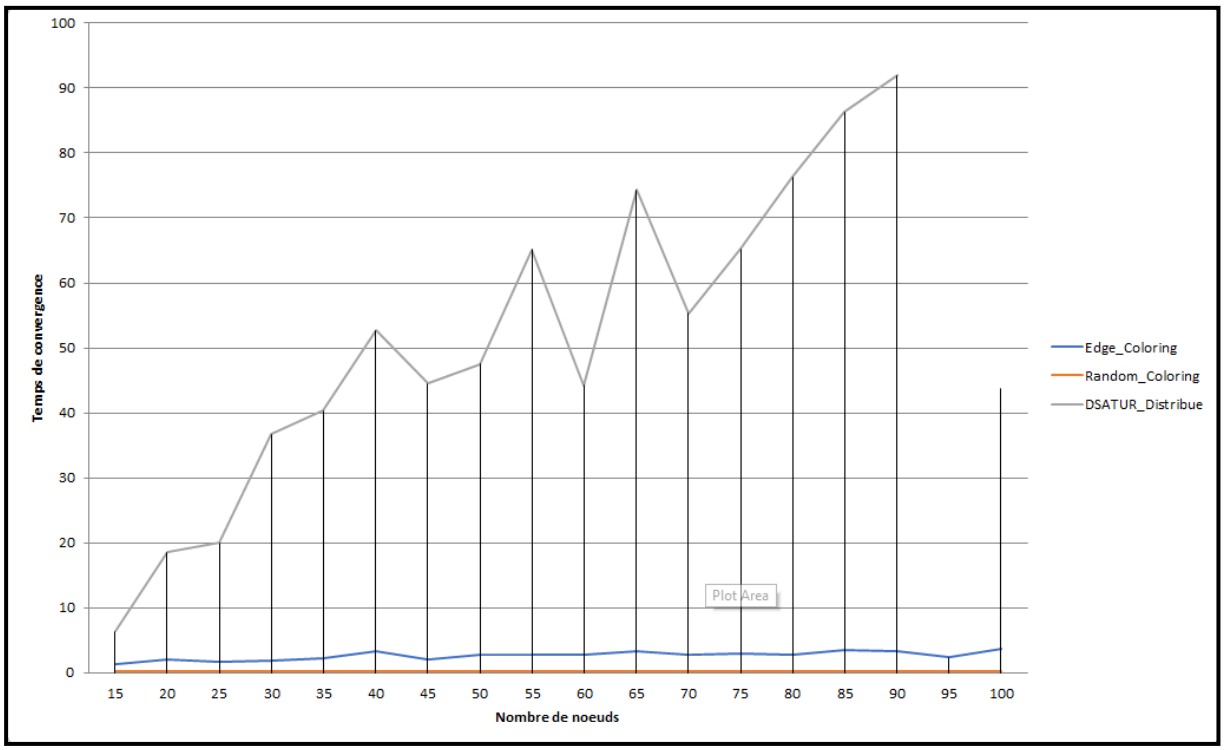

Figure 6.21 Temps de convergence dans un réseau moyen.

6.5.2 Analyse d'un réseau moyen

Nous constatons que le nombre de conflits présentés par l'algorithme DSATUR_DISTRIBUE est beaucoup plus faible que les nombre présentés par les autres algorithmes (presque nul), et que l'algorithme Edge_Coloring est bien meilleur que l'algorithme aléatoire en terme de nombre de conflits.

Nous constatons également que l'algorithme Edge_Coloring a un nombre de message et un temps de calculs significativement plus faibles que ceux de l'algorithme DSATUR_DISTRIBUE. En effet, les performances en termes de nombre de messages et de temps de convergence de l'algorithme Edge_Coloring se rapprochent de ceux de l'algorithme aléatoire qui s'exécute en deux étapes, ce qui présente un grand avantage pour l'algorithme.

6.5.3 Performances d'un réseau dense

Nous présentons dans ce qui suit les courbes du nombre de conflits, du nombre de message et du temps en fonction du nombre de nœuds dans un réseau dense.

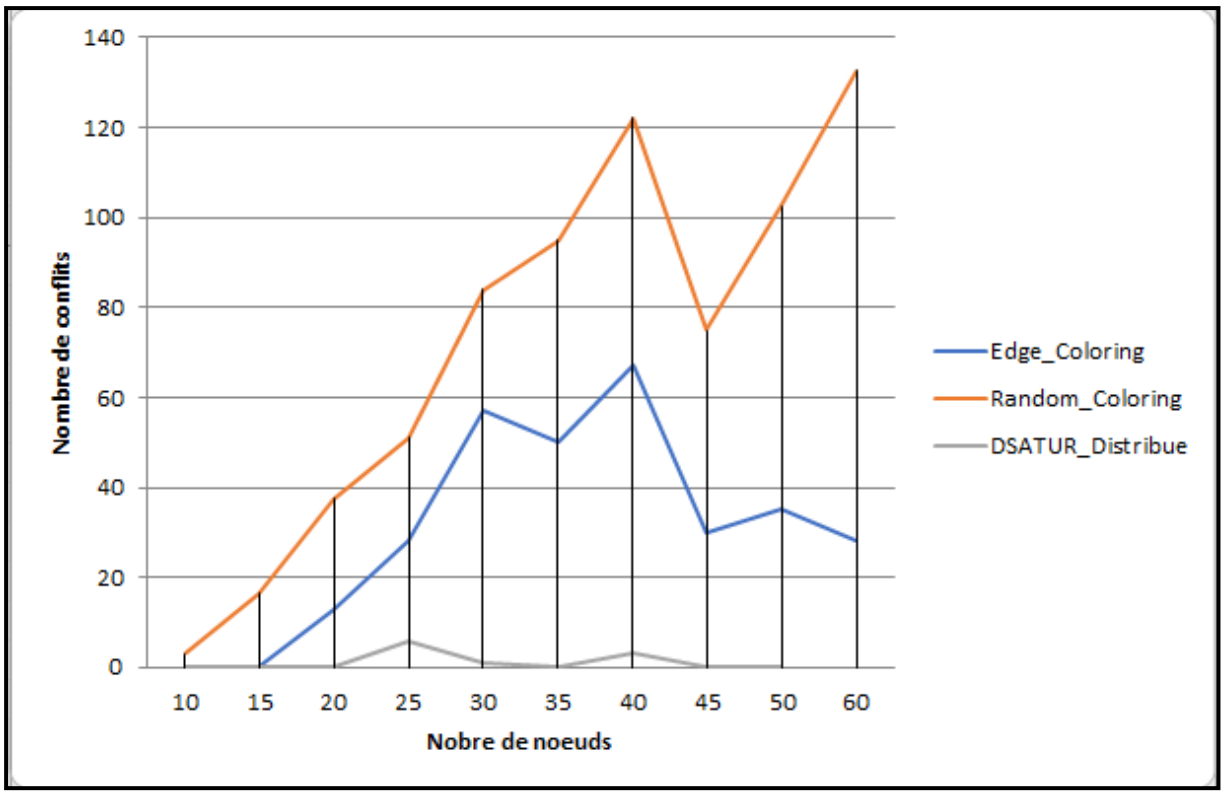

Figure 6.22 Nombre de conflit dans un réseau dense.

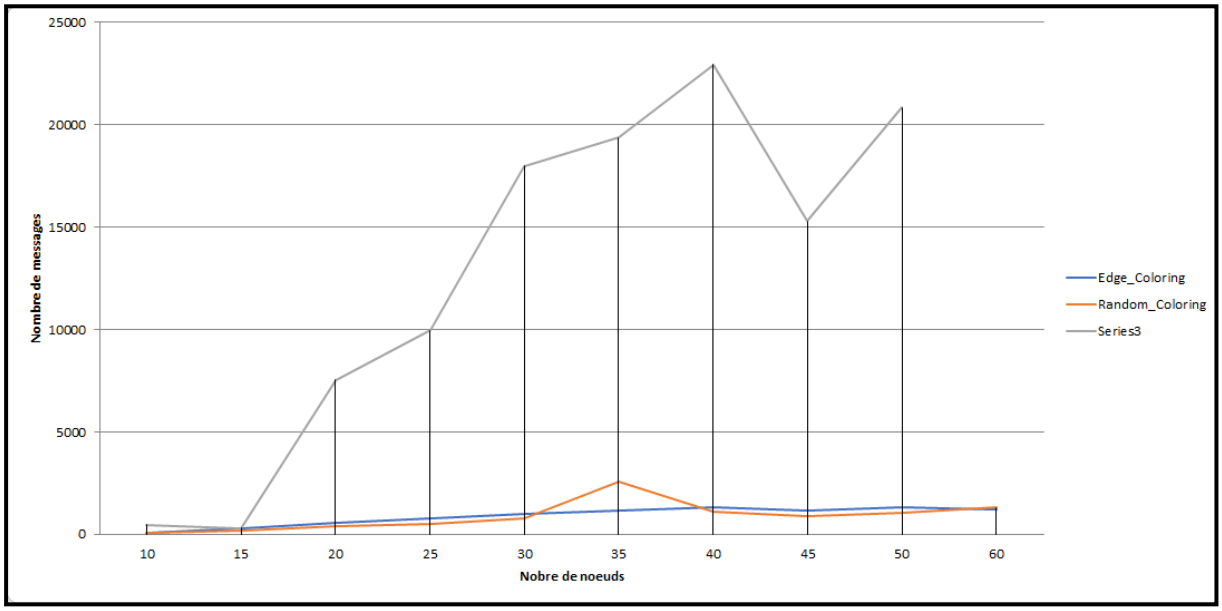

Figure 6.23 Nombre de messages dans un réseau dense.

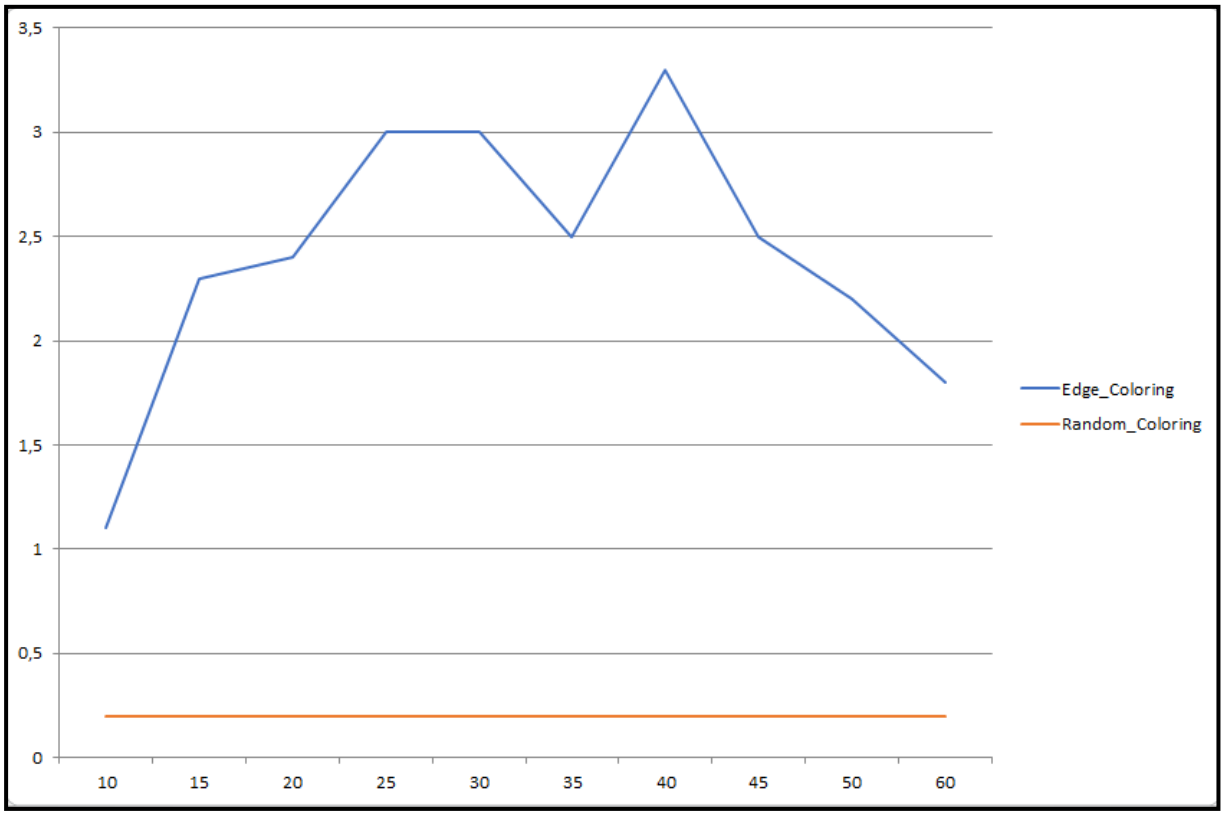

Figure 6.24 Temps de convergence dans un réseau dense.

6.5.4 Performances d'un réseau très dense

Nous présentons dans ce qui suit les courbes du nombre de conflits, du nombre de message et du temps en fonction du nombre de nœuds dans un réseau très dense.

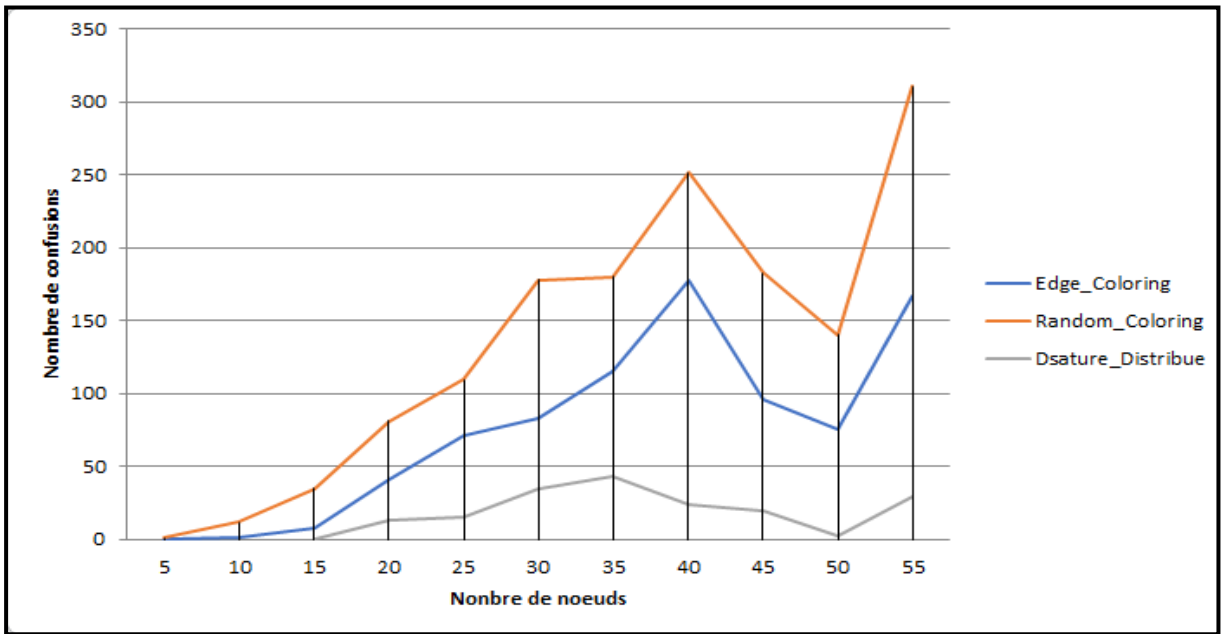

Figure 6.25 Nombre de conflits dans un réseau très dense.

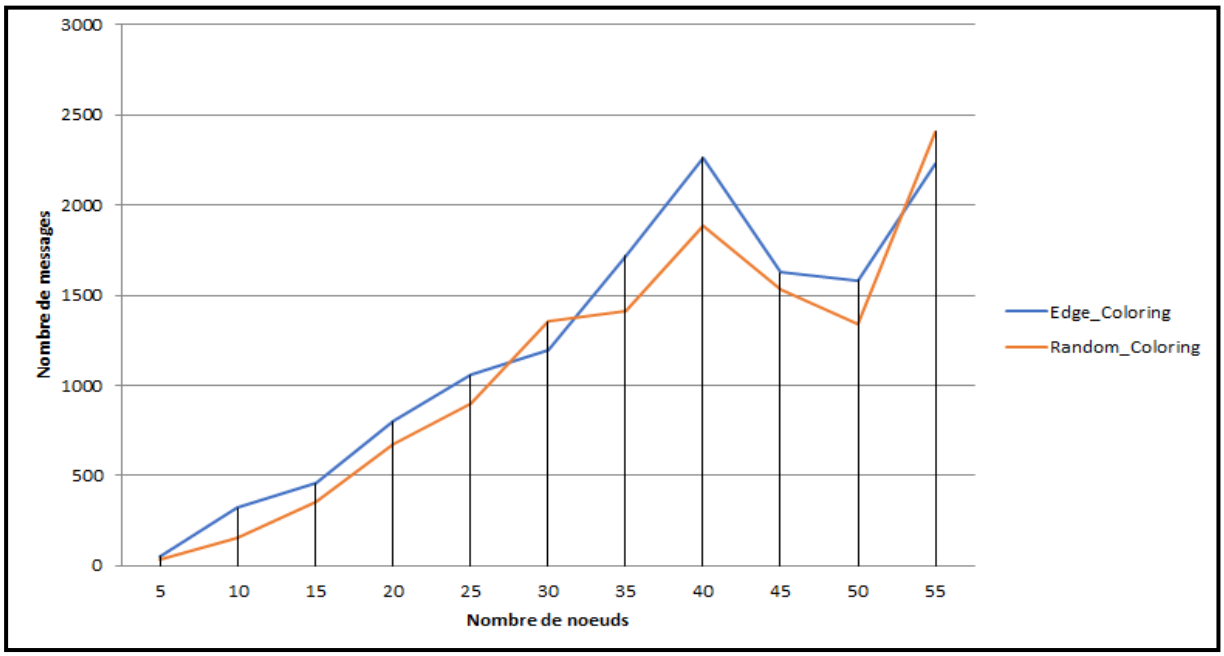

Figure 6.26 Nombre de messages dans un réseau très dense.

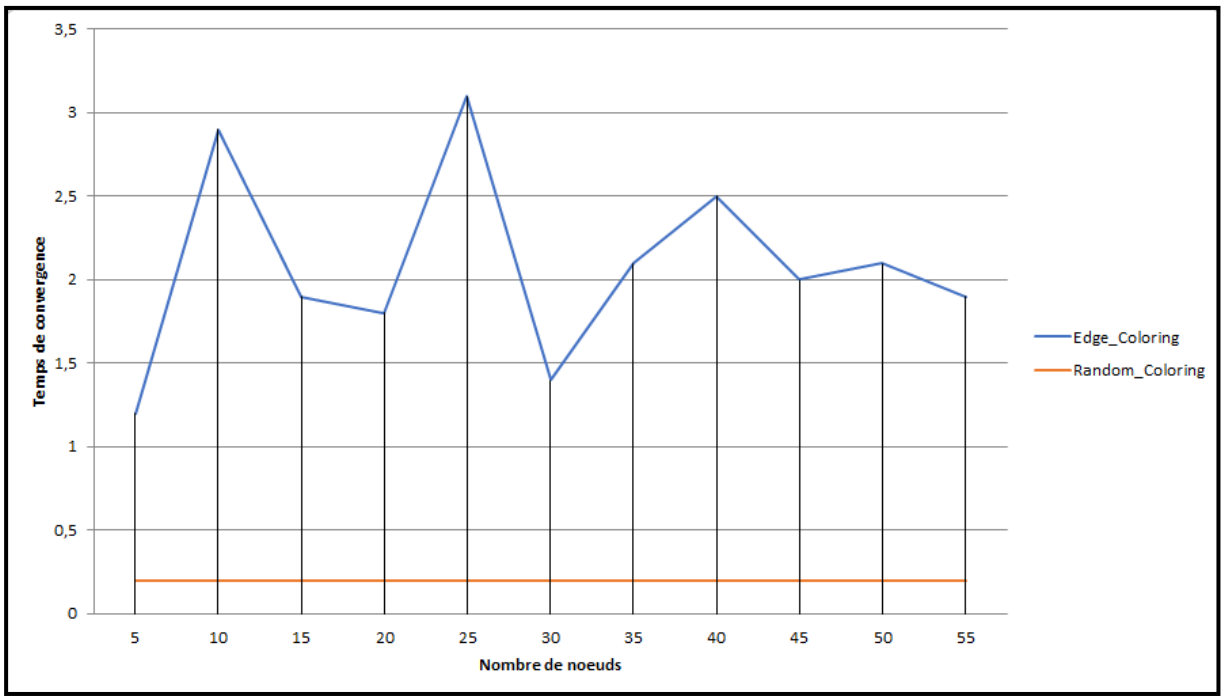

Figure 6.27 Temps de convergence dans un réseau très dense.

6.5.5 Analyse pour un réseau dense

Nous constatons que le nombre de conflits présentés par l'algorithme DSATUR_DISTRIBUE est beaucoup plus faible que les nombre présentés par les autres algorithmes (presque nul), ses performances en terme de conflits restent donc inchangés, tandis que les deux autres algorithmes présentent beaucoup plus de conflits dans un réseau dense. En effets, l'algorithme Edge_Coloring a besoin de plus de couleur pour éviter les conflits mais est bien meilleur que l'algorithme aléatoire.

Pour les performances en nombre de messages et en temps, l'algorithme DSATUR_DISTRIBUE a besoin de beaucoup plus de temps dans un réseau dense pour converger, tandis que l'algorithme Edge_Coloring garde sa caractéristique en terme de nombre de messages qui est proche de celle de l'algorithme aléatoire avec de faibles valeurs.

6.5.6 Conclusion

D'après l'analyse des performances des trois algorithmes, nous pouvons conclure que :
_ Les deux algorithmes (DSATUR et Edge_coloring) sont bien meilleurs qu'un coloriage aléatoire.
_ L'algorithme DSATUR présente un nombre de conflits négligeable et a besoin donc d'un nombre plus faible de couleurs pour fonctionner correctement. Ce constat montre que l'idée

de colorier les nœuds ayant le plus grand degré de saturation en premier donne bien un avantage.

_ L'algorithme de Edge_Coloring distribué à vue locale utilise un nombre de message qui n'est pas très élevé (s'approche du coloriage aléatoire), c'est donc un algorithme qui serait intéressant pour un réseau ad-hoc où le temps de propagation est élevé.

_ L'algorithme DSATUR montre ses limites en termes de nombre de messages et surtout en temps de convergence à cause du mécanisme « impass » qui revient vers les nœuds qui ne sont pas encore coloriés, il est donc moins intéressant pour un réseau ad-hoc large.

7 Annexe A

7.1 Algorithme Edge_DSATUR

L'algorithme Edge_DSATUR est un algorithme distribué de coloriage d'arêtes. Il est constitué de deux phases. L'idée de cette approche est d'étudier les performances d'un algorithme de coloriage d'arêtes en passant par un coloriage de nœuds. En effet, la phase 1 de l'algorithme consiste en un coloriage des nœuds du graphe grâce aux échanges de messages entre les voisins directs. Le coloriage des nœuds est basé sur le degré de saturation (DSAT). La deuxième phase débute une fois la première phase terminée. Elle consiste en un coloriage des arêtes du graphe. Le nœud ayant la plus petite couleur de son voisinage procède au coloriage de toutes ses arêtes.

7.2 Pseudo-code

PHASE 1

Hypothèses :
_ Liens de communications fiables et bidirectionnels.
_ Réseau connexe.

Wait()
{
Debut
    break ;
Fin ;
}

Coloring()
{
Debut
    Permission ++;
    If (Permission == Card (Voisin i))
        Color i = vrai;
        Mycolor i = c tel que c ∈ Color_Palette & ∀ m ∈ Color_Palette : c < m ;
        ∀ k ∈ voisin i envoyer (CalculDsatMsg(Mycolor i)) à Pk.
    Fin ;
Fin ;
}

```
DsatColoring()
{
Debut
    DsatPermission=0;
    ∀ k ∈ voisin i :
        if ( Dsat i < Dsatvoisin i [k]) alors
            Wait ();
        elseif  ( Dsat i > Dsatvoisin i [k]) alors
            DsatPermission i ++;
        elseif (Dsat i == Dsatvoisin i [k]) alors
            if ( i > ID (k)) alors
                DsatPermission i ++;
            Fin;
        Fin;

    If ( DsatPermission == Card (Dsatvoisin i) ) alors
        {
        Color i = vrai;
        Mycolor i = c  tel que c ∈ Color_Palette & ∀ m ∈ Color_Palette : c < m ;
        ∀ k ∈ voisin i  envoyer (CalculDsatMsg(Mycolor i)) à Pk.
        }
    Fin;
Fin;
}
```

Contexte local à Pi :
Voisin i : ensemble initialisé aux voisins de Pi;
Color i: booléen;
Mycolor i: entier;
Permission: entier;
Degre i: entier;
Dsat i: entier;
Color_Palette i : ensemble initialisé à l'ensemble des couleurs disponibles;
Dsatvoisin i: ensemble initialisé aux Dsat des voisins de i, initialement 0;

DsatPermission i: entier ;

Messages :
DegreMsg() ;
CalculDsatMsg() ;
DsatMsg() ;

Initialisation :

```
Debut
|
|       Color i = faux ;
|       ∀ k ∈ voisin i  envoyer (DegréMsg(Degre i)) à Pk.
|
|
Fin ;
```

Lors de la réception de DegreMsg(Degre j) de Pj :

```
Debut
|
|       If (Degre i > Degre j)  alors
|       |
|       |       Coloring ();
|       Else if ( Degre i < Degre j )  alors
|       |
|       |       Wait();
|       Else
|       |       If ( i > j)  alors
|       |       |
|       |       |       Coloring();
|       |
|       |       Fin;
|       Fin;
|
Fin;
```

Lors de la réception de CalculDsatMsg(Mycolor j) de Pj :

```
Debut
    If (Color i = faux) alors
    {
        Dsat i ++;
        Dsat j= 0 in Dsatvoisin i  ;
        Color_Palette i = Color_Palette i – {Mycolor j}
        ∀ k ∈ voisin i  envoyer (DsatMsg(Dsat i)) à Pk.
        DsatColoring() ;
    }
    Fin ;
Fin ;
```

Lors de la réception de DsatMsg(Dsat j) de Pj :

```
Debut
    MAJ (Dsatvoisin i)
Fin;
```

PHASE 2

Hypothèse :

_ Terminaison de la phase 1.

Contexte local à Pi :

CouleurHistorique i : ensemble des couleurs déjà attribuées aux arêtes adjacentes.
Couleur i : ensemble de couleurs disponibles du nœud Pi.
CouleurVoisin i [voisin i]: liste d'ensemble de couleurs disponibles dans chaque voisin.

Coloriage des arêtes :

Le nœud ayant la plus petite couleur sur le lien procède au coloriage de l'arête E tel que :
Soit u, v deux nœuds partageant la même arête E tel que Couleur (u) < Couleur (v).

If ((Couleur i ∩ CouleurVoisin u (v)) ≠ CouleurHistorique i) alors
{
Couleur(E)= Min (Couleur i ∩ CouleurVoisin (v)) ;
MAJ (CouleurHistorique i);
}
Else Couleur(E)= rand (Couleur i ∩ CouleurVoisin (v)) ;

Fin ;

7.3 Réalisation

La phase 1 de l'algorithme a bien été implémentée avec l'outil OMNET++. L'implémentation de la phase 2 n'a pas abouti.

7.4 Déroulement de l'algorithme Edge_DSATUR

Dans cette partie, nous procédons au déroulement de la phase 1 de l'algorithme, qui a été bien implémenté sur OMNET++, sur la topologie illustré par la figure 7.1.

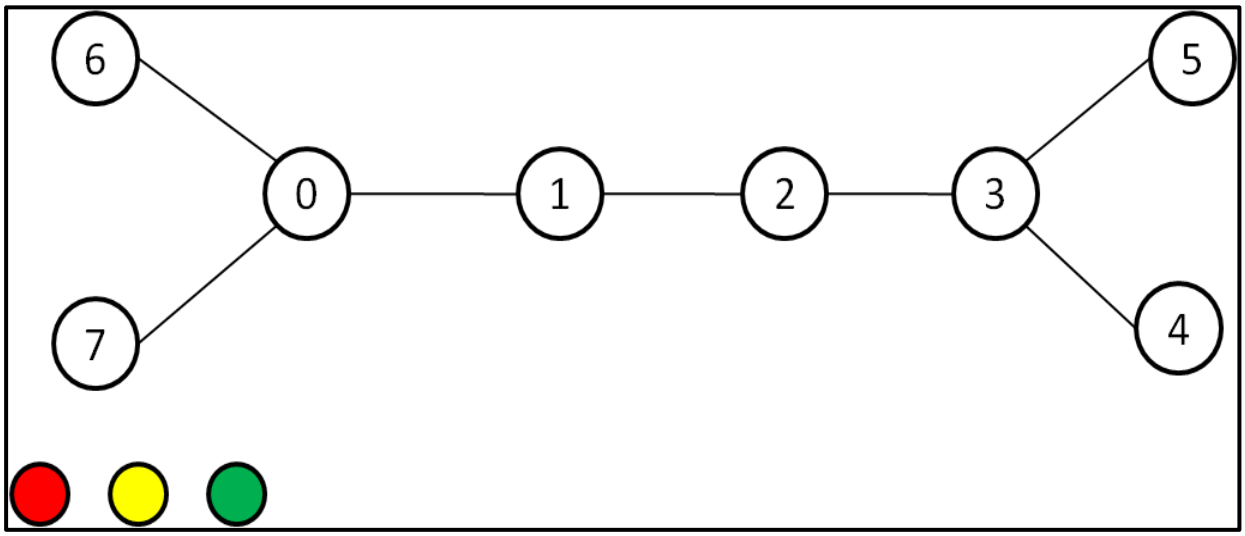

Figure 7.1: Topologie du déroulement de l'algorithme Edge_DSATUR.

_ Lors de la phase d'initialisation, chaque nœud envoie son degré à tous ces voisins directs.

_ A la réception du message < DegreMsg > encapsulant le degré du nœud source, un nœud se colorie, avec la plus petite couleur, s'il a le plus grand degré parmi tous ses voisins. La figure 7.2 illustre cette étape.

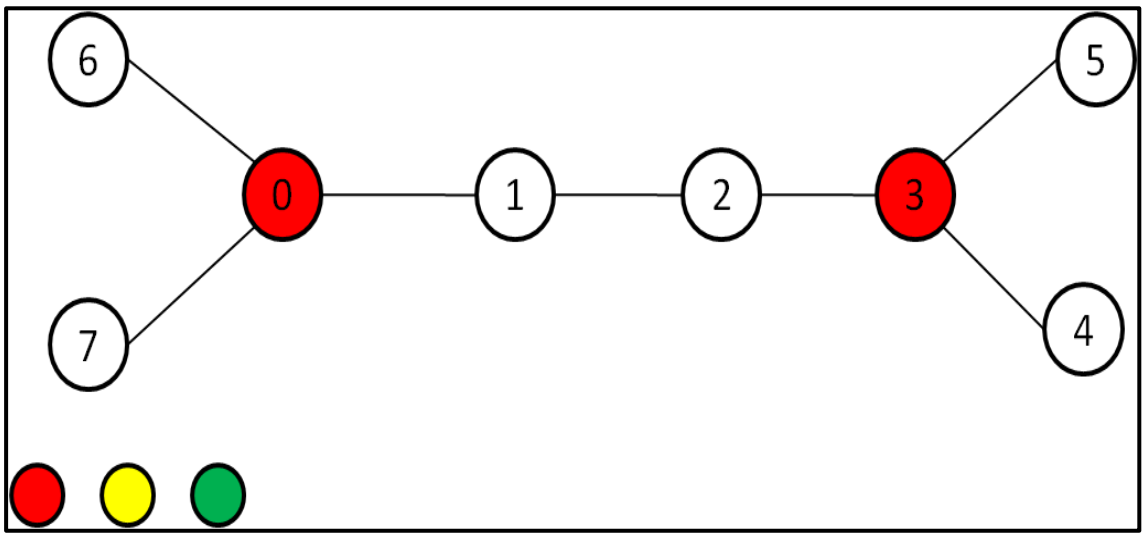

Figure 7.2: Round 1 du coloriage des nœuds.

_ Lorsqu'un nœud se colorie, il envoie sa couleur aux voisins directs et leur demande de calculer leur degré de saturation.

_ Lors de la réception du message < CalculDsatMsg> encapsulant la couleur du nœud source, un nœud qui ne s'est pas encore colorié, met à jour son degré de saturation et l'envoie à ses voisins directs. Si le nœud a le plus grand degré de saturation parmi tous ses voisins alors il se colorie avec la plus petite couleur disponible. Si deux nœuds voisins ont le même degré de saturation le nœud qui a le plus grand ID se colorie en premier.

_ Lors de la réception du message < DsatMsg> encapsulant le degré de saturation du nœud source, un nœud met à jour sa liste des degrés de saturation de ses voisins. Les figures 7.3 et 7.4 illustre le coloriage des nœuds suite à la mise à jour des degrés de saturation des nœuds.

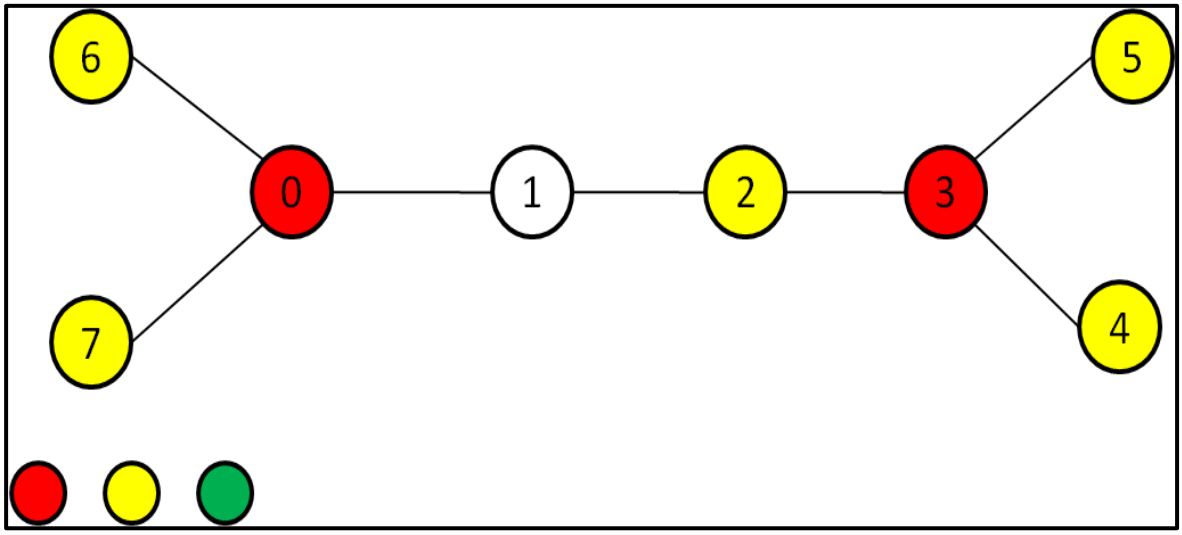

Figure 7.3: Round 2 du coloriage des nœuds.

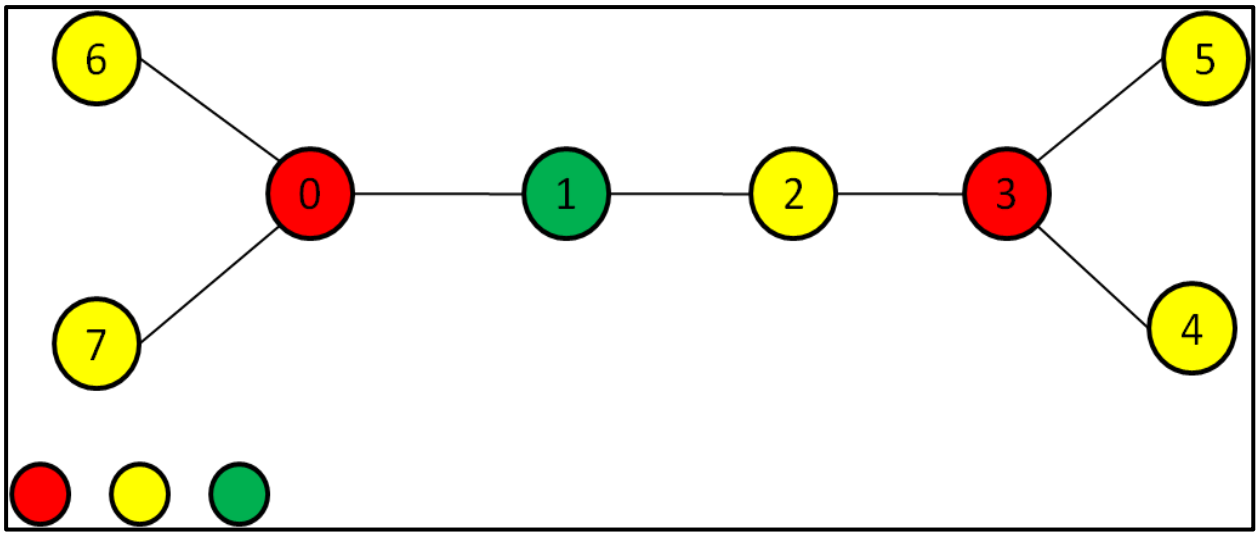

Figure 7.4: Round 3 du coloriage des nœuds.

La figure 7.5 représente le résultat de l'exécution de la phase 1 de l'algorithme Edge_DSATUR sur OMNET++.

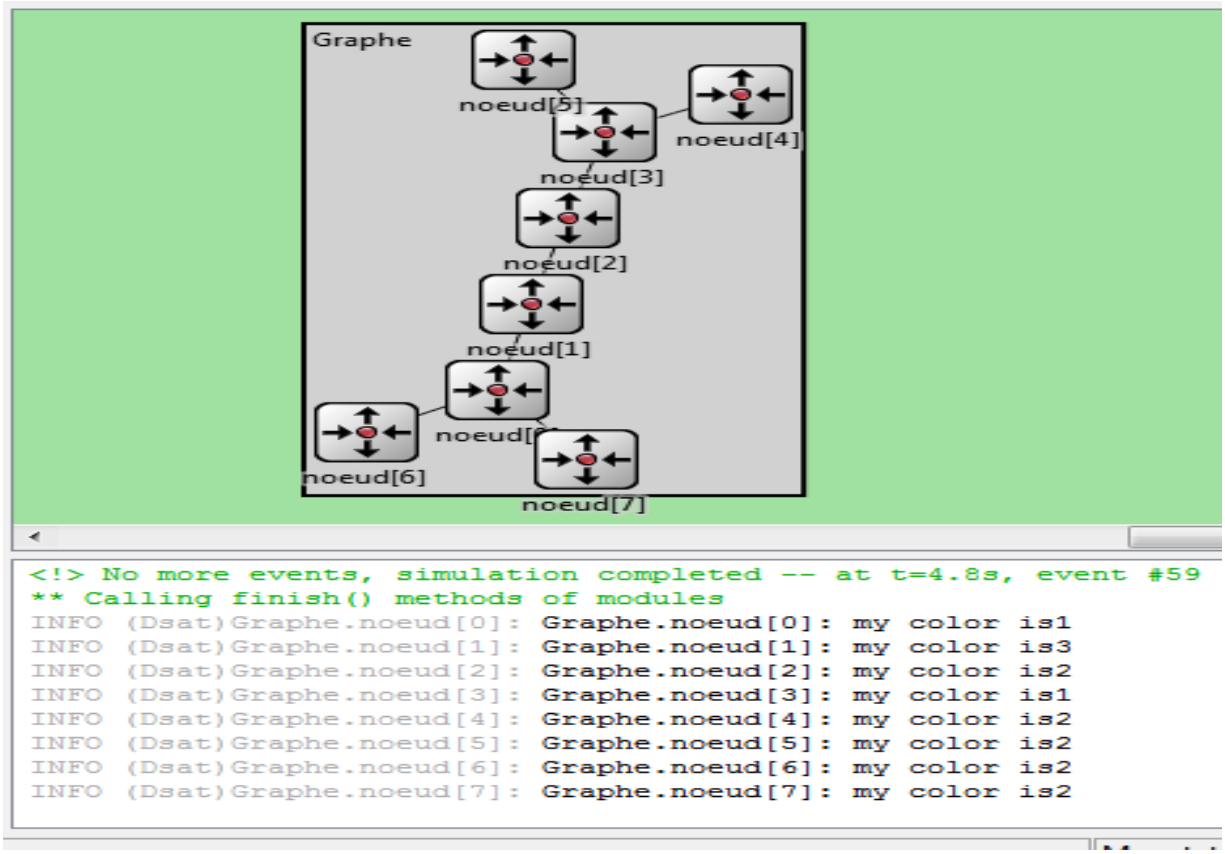

Figure 7.5: Résultat d'exécution de la phase 1 de l'algorithme Edge_DSATUR sur OMNET++.

8 Annexe B

Dans cette annexe, nous présentons une partie du code source qui traite les messages < already_colored > et <color_yourself>. La figure 8.1 illustre le code source.

```
    }
    if (strcmp(msg->getName(),"my color")==0){
        MyColor *myColor_msg = check_and_cast<MyColor *>(msg);
        list_color.erase(std::remove(list_color.begin(),list_color.end(),myColor_msg->getColor()),list_color.end());
        actif_neighbors.erase(std::remove(actif_neighbors.begin(),actif_neighbors.end(),myColor_msg->getSource()),actif_neighbors.end());
        remain.erase(std::remove(remain.begin(),remain.end(),myColor_msg->getSource()),remain.end());
        degSat+=1;
    }
    if (strcmp(msg->getName(),"color_yourself")==0){    // Je recoit un message color_yourself
        Color_yourself *colorYourself_msg = check_and_cast<Color_yourself *>(msg);
        if(colored == false){   //si je n'ai pas encore de couleur (pas de probleme de ds request par noeud "m" avant color yourself par noeud "n")
            colored = true;     //car je vais prendre une couleur
            master = colorYourself_msg->getArrivalGate()->getIndex();   //me servira pour passer l'impass s'il y en aura
            std::sort(list_color.begin(),list_color.end());             //pour prendre la plus petite couleur
            if (list_color.size()!=0) color = list_color[0];            //si j'ai des couleur je prend
            else {
                color = 1;
                Conflit *conflit_msg = generateMessage7();              // si je n'ai pas de couleur j'informe mon voisin qu'il y a un conflit
                send(conflit_msg, "gate$o",master);
            }
```

```
            list_color.erase(std::remove(list_color.begin(),list_color.end(),color),list_color.end());  //je supprime la couleur de la liste
            for (int i=0; i < colorYourself_msg->getRemainArraySize(); i++) remain.push_back(colorYourself_msg->getRemain(i));
            EV<<"Le lien "<<getIndex()<<" prend la couleur C"<<color<<". \n";

            MyColor *color_msg = generateMessage2();        //j'informe mes voisins
            for (int i = 0; i < gateSize("gate$o"); i++)
            {
                MyColor *copy = color_msg->dup();
                send(copy, "gate$o", i);
            }
            EV<<"Le noeud virtuel "<<getIndex()<<" informe ses voisins qu'il a pris la couleur C"<<color<<". \n";

            if(actif_neighbors.size() !=0){
                EV<<"Le noeud "<<getIndex()<<" demande les DS de ses voisins: ";
                DsRequest *dsRequest_msg = generateMessage4();
                for (auto& i:actif_neighbors)       //je demande les DS de mes voisins actifs si j'en ai
                {
                    EV<<i<<", ";
                    DsRequest *copy = dsRequest_msg->dup();
                    send(copy, "gate$o", u[i]);
                }
                EV<<"\n";}
            else if(actif_neighbors.size() == 0 && colorYourself_msg->getRemainArraySize() !=0){
                EV<<"Le noeud "<<getIndex()<<" n'a plus de voisin actif mais il reste des noeuds non colories (renvoi du msg impass). \n";
                Impass *impass_msg = generateMessage6();        //je renvoi impass si j'en ai pas de voisins actifs mais reste des noeuds dans le graphe
                send(impass_msg,"gate$o",master);
                }
            }
         else{          //else du premier if (ligne 172)
            AlreadyColored *already_msg = generateMessage8();       //si je suis deja colorie (probleme du ds request avant color yourself)
            send(already_msg,"gate$o",colorYourself_msg->getArrivalGate()->getIndex());
        }
```

```
    }
    else if (strcmp(msg->getName(),"already colored")==0){  //Reception d'un message already colored
        AlreadyColored *already_msg = check_and_cast<AlreadyColored *>(msg);    //supprimer le noeud des voisins actifs
        actif_neighbors.erase(std::remove(actif_neighbors.begin(),actif_neighbors.end(),already_msg->getSource()),actif_neighbors.end());
        if(already_msg->getColor() == color){   //Si mon voisin est colorié avec la meme couleur que moi (probleme de ds request avant color yourself)
            if(list_color.size()!=0){       // Si j'ai d'autre couleurs disponibles
                color = list_color[0];      //Je change ma couleur pour ne plus etre en conflit avec mon voisin
                MyColor *color_msg = generateMessage2();    //J'informe mes voisins
                for (int i = 0; i < gateSize("gate$o"); i++)
                {
                    MyColor *copy = color_msg->dup();
                    send(copy, "gate$o", i);
                }
                EV<<"Le noeud virtuel "<<getIndex()<<" informe ses voisins qu'il change sa couleur en C"<<color<<". \n";
            }

        }
```

Figure 8.1: Implémentation des messages <color_yourself> et <already_colored>

9 Références

[1] Z. Tabakovic and M. Grgic, « Cognitive radio frequency assignment with interference weighting and categorization, » EURASIP Journal on Wireless Communications and Networking, 2016, doi: 10.1186/s13638-016-0536-1.

[2] F. Richard Yu, «Cognitive Radio Mobile Ad Hoc Networks,» Springer Science+Business Media, LLC, 2011, doi : 10.1007/978-1-4419-6172-3.

[3] A.Khattab, D.Perkins and M. Bayoumi, « Cognitive Radio Networks From Theory to Practice,» Springer Science+Business Media New York, 2013, doi : 10.1007/978-1-4614-4033-8.

[4] S. Gupta and V. Malagar, « IEEE 802.22 Standard for Regional Area Networks, » 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS), 2017, doi: 10.1109/ICNGCIS.2017.20.

[5] R.M.R. Lewis, « A Guide to Graph Colouring Algorithms and Applications,» Springer International Publishing Switzerland, 2016, doi : 10.1007/978-3-319-25730-3.

[6] H. Harianto and Drs.M.M. Kom, « Implementation Coloring Graph and Determination Waiting Time Using Welch-Powell Algorithm in Traffic Light Matraman, » 3rd International Conference On Research, IMPLEMENTATION AND EDUCATION OF MATHEMATICS ANS SCIENCE, pp. 155-160, May.2016.

[7] M. K. Aguilera, « Distributed computing, » Springer, 26th International Symposium, Disc, Salvador, Brazil, October 16-18. 2012.

[8] https://doc.omnetpp.org/omnetpp/manual/