2

Concepts de base relatifs aux RCSFs

Nous enchainerons les définitions du chapitre précédent avec tous les concepts de base constituant un jargon propre aux réseaux de capteurs sans fil qui vont faire l'objet de ce chapitre.

2.1 Définition des RCSFs

Un réseau de capteurs sans fil est un réseau Ad-hoc [1] composé d'un grand nombre de capteurs sans fil connectés entre eux, placés dans des positions prédéterminées ou déployés aléatoirement dans une zone géographique appelée champs de capture.

Chaque capteur a la possibilité de collecter des informations de tout type (Température, taux d'humidité, signal sonore, niveau d'eau, etc). Les données sont routées de capteurs en capteurs jusqu'à ce qu'elles atteignent la station de base. C'est-à-dire que des chemins sont sélectionnés dans le réseau pour acheminer les données collectées par un capteur vers le *sink*.

2.2 Composantes des capteurs sans fil

Les capteurs sans fil sont composés de :

- **Une unité d'acquisition** (ou de capture) : qui comporte un ou plusieurs capteurs qui se chargent de recueillir des données se trouvant à proximité. Ces données sont ensuite converties en une valeur numérique grâce à un convertisseur analogique/numérique.
- Une unité de traitement : qui est composée d'un processeur et d'une mémoire, les données capturées lui parviennent de l'unité d'acquisition pour être analysées. Elle les envoie par la suite à l'unité de communication.
- **Une unité de communication** : qui est responsable de toutes les émissions et réceptions des données via un support un support de communication radio.
- **Une unité d'énergie** : Munie d'une batterie, elle alimente toutes les autres unités. L'énergie est la ressource la plus précieuse étant donné qu'elle influe sur la durée de vie du capteur.

1. Les réseaux Ad-hoc sont des réseaux sans fil capables de s'organiser sans infrastructure définie au préalable

Station de base (Sink)

FIGURE 2.1 – Réseau de capteurs sans fil

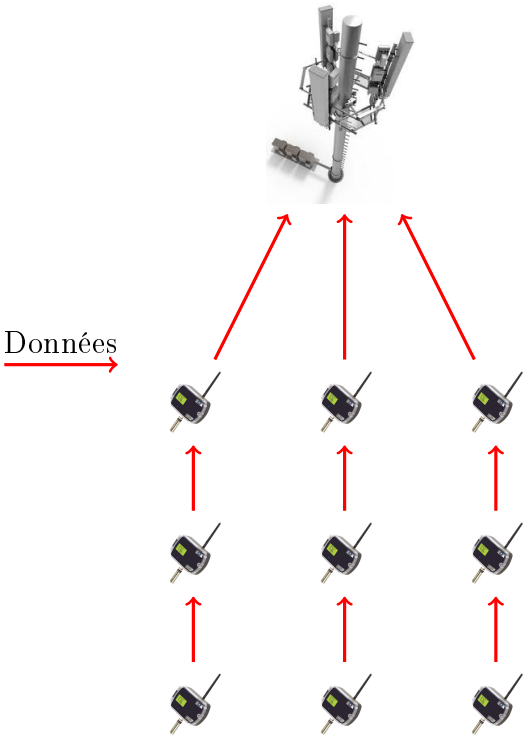


FIGURE 2.2 – Routage des données dans un RCSF

Ces composantes sont représentées par le schéma de la figure 2.3.

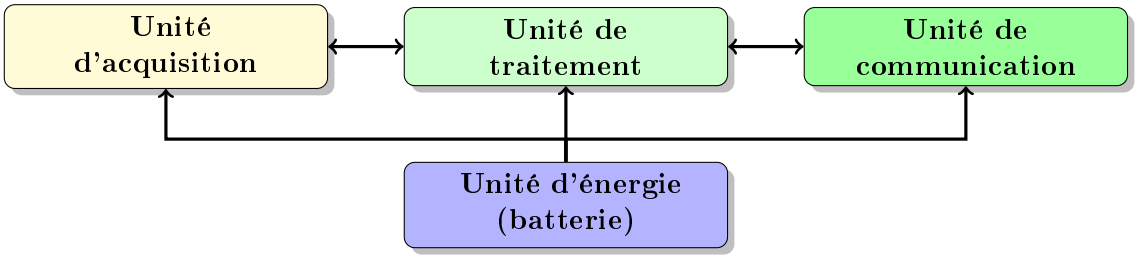


FIGURE 2.3 – Composantes d'un capteur sans fil

2.3 Modélisation des RCSFs par les graphes

Les RCSFs sont représentés par un graphe $G = (V, E)$ où V est l'ensemble des sommets sont les capteurs et le nœuds puits, et E est l'ensemble des arêtes tel que, une arête existe entre deux sommets si un des deux capteurs les représentants (ou entre un capteur et le *sink*) se trouve dans la plage de transmission de l'autre.

Dans la figure 2.4, les capteurs (numérotés de 1 à 9) et la station de base (*sink*) représentent les sommets du graphe. Le cercle en bleu représente la plage de transmission du sommet (capteur) 5, il peut donc communiquer (émettre et recevoir) des données à tous les capteurs se trouvant à l'intérieur dudit cercle.

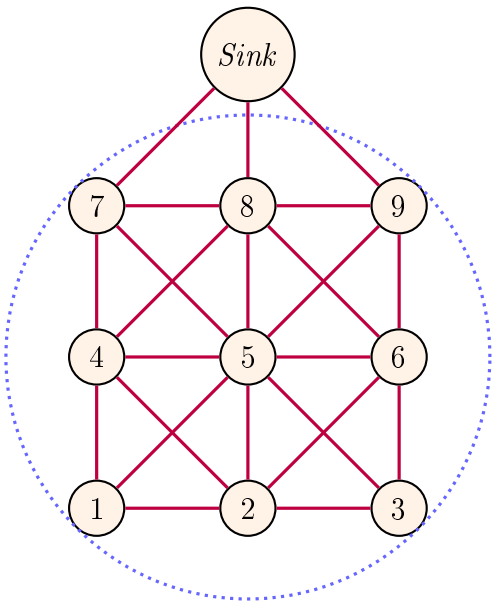


FIGURE 2.4 – Exemple de graphe représentant un RCSF

Suite au déploiement des capteurs, le RCSF obtenu, peut avoir une structure particulière de graphe telle que les arbres et ses cas particuliers, les mailles... Comme le montre la figure 2.5.

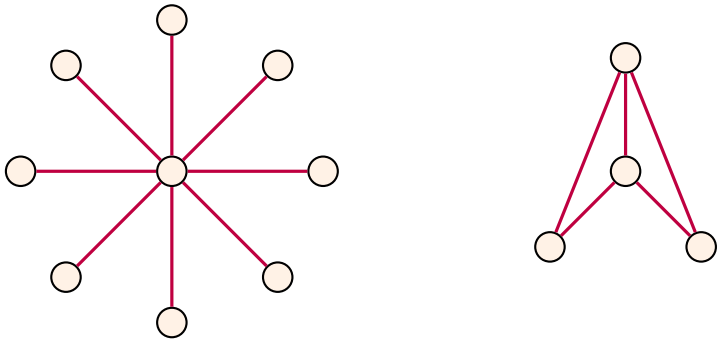


FIGURE 2.5 – Structure en étoile et structure en maille

2.4 Domaines d'application des RCSF

Dans un RCSF, les nœuds capteurs sont capables de s'organiser en réseau et de se contrôler et de se gérer efficacement sans aucune intervention. Cette particularité facilite le déploiement des capteurs sur un territoire, sur ou sous terre et même sous l'eau, et fait bénéficier plusieurs domaines des RCSFs, notamment :

- **Le domaine militaire** : surveillance des forces amies, l'équipement et les munitions, surveillance du champ de bataille, identification des forces ennemies et du terrain, évaluation des dégâts causés par une bataille, détection d'attaque nucléaire, biologique et chimique...
- **L'environnement** : détection des feux de forêts et d'inondations, et dans l'agriculture (surveiller le niveau de pesticides dans l'eau potable, le niveau d'érosion du sol et le niveau de pollution de l'air en temps réel)...
- **Le domaine de la santé** : surveillance de la tension artérielle, de la température et du rythme cardiaque des patients, administration des médicaments dans les hôpitaux (détection d'allergies)...
- **La domotique** : automatisation des maisons en permettant aux utilisateurs de gérer à distance l'électroménager et les appareils. Cela entre dans la conception de villes intelligentes (*smart cities*)

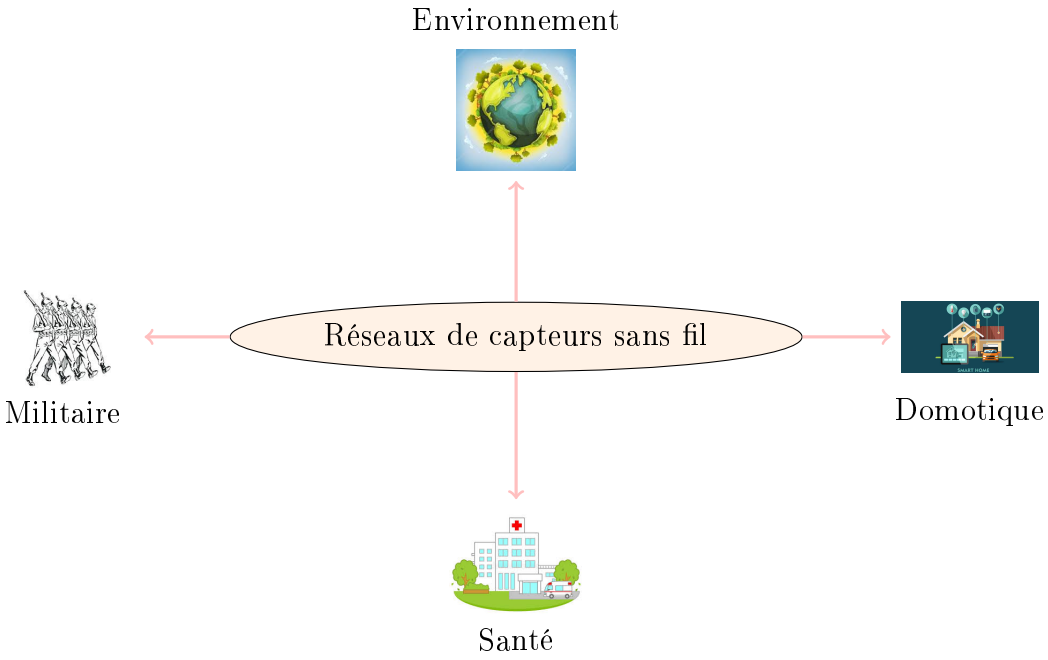


FIGURE 2.6 – Domaines d'application des RCSFs

2.5 Agrégation des données

2.5.1 Définition et objectifs

L'une des principales préoccupation de l'agrégation des données et de minimiser la consommation de l'énergie des capteurs et de prolonger par conséquent la durée de vie du réseaux. Agréger des données dans un RCSF est l'action de combiner des données provenant de différents nœuds capteurs pour les faire parvenir au *sink*, de sorte à :

- Éviter les **transmissions redondantes** dans le réseau étant donné que la communication est de loin la première source de consommation de l'énergie.
- Assurer une certaine **précision des données** lors de leur regroupement en évitant de perdre les informations utiles.
- **Diminuer les risques de congestion** dans le réseau.

Toutefois, dans certaines structures de réseaux, l'agrégation des données peut facilement devenir contraignant, voire handicapant, comme dans le cas de la structure en étoile (Figure 2.5). Si le point d'articulation[2] de l'étoile représente un capteur, alors tous les autres capteurs devront transmettre leurs données à ce sommet, qui se chargera de les communiquer au *sink*. Si ce capteur cesse de fonctionner, pour une quelconque raison, ceci altérera tout le réseau.

2.5.2 Notions sur la latence

Plusieurs protocoles d'agrégation des données ont été proposés afin de minimiser l'énergie consommée. Cependant, ils mènent à une augmentation de ce qu'on appelle la latence.

2. Un point d'articulation dans un graphe est un sommet qui déconnecte le graphe si on le retire.

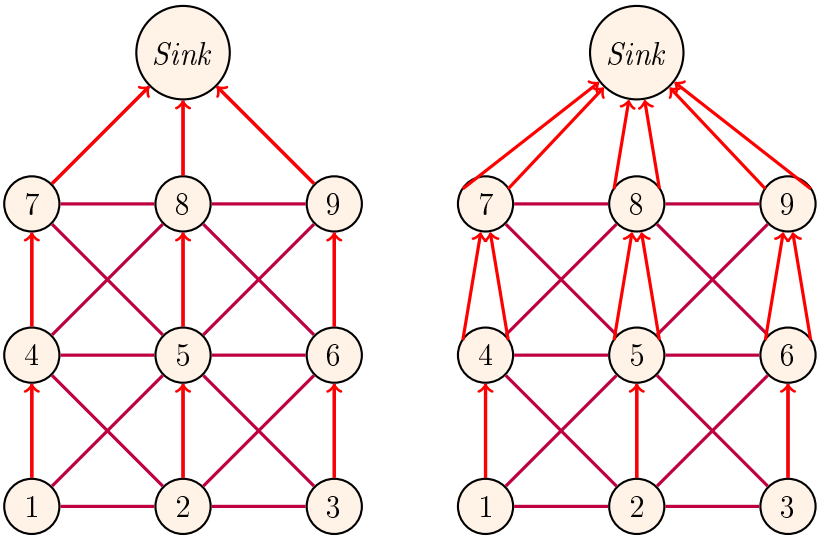


FIGURE 2.7 – Acheminement avec et sans agrégation

Définition

On appelle latence, le nombre de périodes de temps (*slot* en anglais) nécessaires à un ensemble de données pour arriver à la station de base, puisque chaque nœud possède un délai d'attente avant de recevoir des données de ses voisins.

Il faut alors agréger les données, déterminer à chaque période de temps les transmissions qui peuvent se faire (entre un capteur et un autre) et veiller à réduire le nombre total de périodes. Ceci revient alors à trouver une planification optimale d'agrégation des données qui minimise le temps de latence.

2.6 Description des principales contraintes du problème [Yagouni et al., 2015]

Soit une application de RCSF définie par un graphe $G = (V, E)$. Le problème qui consiste à trouver une planification optimale d'agrégation des données qui minimise le temps de latence doit impérativement satisfaire aux conditions et contraintes suivantes :

- Chaque nœud a un identifiant unique (un chiffre ou une lettre qui le représente).
- Chaque nœud se situe de manière précise dans le temps, il ne peut transmettre des données que durant la période où il est autorisé à le faire.
- Il ne peut y avoir une transmission partielle des données (la transmission d'un nœud à un autre se fait entièrement durant une seule période de temps).
- Toute période dure exactement une unité de temps (*slot*).
- Le temps nécessaire dédié au traitement des données est négligeable.
- L'agrégation est réalisé durant la période de temps où un nœud reçoit une transmission.
- Chaque nœud ne doit transmettre que durant une et une seule période de temps.
- Contraintes de précédence : Pour transmettre des données au nœud k, le nœud i doit d'abord les transmettre au nœud j, et ce dernier devra attendre de recevoir

les données de tous ses prédécesseurs et les communiquer à la fois au nœud k. Par conséquent, le noeud j ne peut transmettre et recevoir des données à la même période ($t = t'$) ou avant de recevoir des données de ces prédécesseurs ($t' < t$). (Figure 2.8)

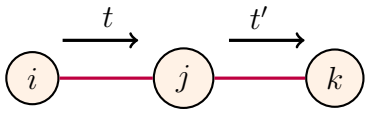


FIGURE 2.8 – Contrainte de précédence

- Contraintes de chevauchement : Un noeud j ne peut recevoir plus d'une transmission à la fois durant une même période t. (Figure 2.9)

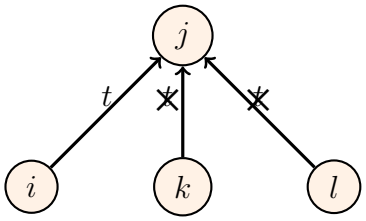


FIGURE 2.9 – Contrainte de chevauchement

- Contraintes de collision : Si le nœud k transmet à son voisin l durant la période t, tout noeud j qui communique avec le nœud k ne peuvent recevoir une transmission durant cette même période t. (Figure 2.10)

FIGURE 2.10 – Contrainte de collision

3 État de l'art

3.1 Introduction

La minimisation de la consommation de l'énergie dans les RCSFs a été largement étudiée, le problème de la minimisation du temps de latence dans la planification de l'agrégation des données a donné naissance à plusieurs protocoles se déroulant presque tous en deux phases. La première phase consiste en la construction **d'arbres d'agrégation**, la seconde phase produit, quant à elle, une planification de **l'agrégation des données**. Dans les sections suivantes, nous résumerons les principaux protocoles existants et nous verrons deux modèles mathématiques du le problème.

3.2 Protocoles existants

3.2.1 Protocole SDA

En 2005, Chen et al ont proposé le protocole Shortest Data Aggregation. Ce protocole est basé sur l'arbre couvrant, il exécute simultanément la construction de l'arbre et la planification d'agrégation des données. L'arbre des plus courts chemins est d'abord crée, pour être utilisé comme un paramètre d'entrée dans l'algorithme de planification.

A chaque itération, l'algorithme produit un planning correspondant à une période de temps et ce en choisissant parmi les feuilles de l'arbre des plus courts chemins, enraciné au *sink*, les nœuds pouvant transmettre au *slot* courant.

Une borne supérieure de la latence a été donnée ($(\Delta-1)R$ où Δ est le degré maximum du réseau et R son rayon) et il a été montré que l'algorithme résout le problème correctement.

La figure (a) représente un RCSF modélisé par un graphe G, la figure (b) est l'arborescence des plus courts chemins enracinée au *sink* obtenue par l'algorithme de Djikstra ou autre. Dans la figure (c) la planification des nœuds est effectuée comme suit :

A la première itération, on relève les feuilles de l'arborescence des plus courts chemins, c'est-à-dire les nœuds $\{1, 3, 5, 6, 7, 9\}$. On trie ces feuilles selon l'ordre décroissant du nombre de voisin dans le graphe G qui ne sont pas des feuilles de l'arborescence. Dans notre cas, on obtient l'ordre suivant : $\{7, 9, 1, 3, 5, 6\}$.

En commençant par le *slot* 1, on examine d'abord le sommet 7 qui a le *sink* et le sommet 4 comme voisins. On va chercher parmi les voisins de 7 celui qui n'a aucune

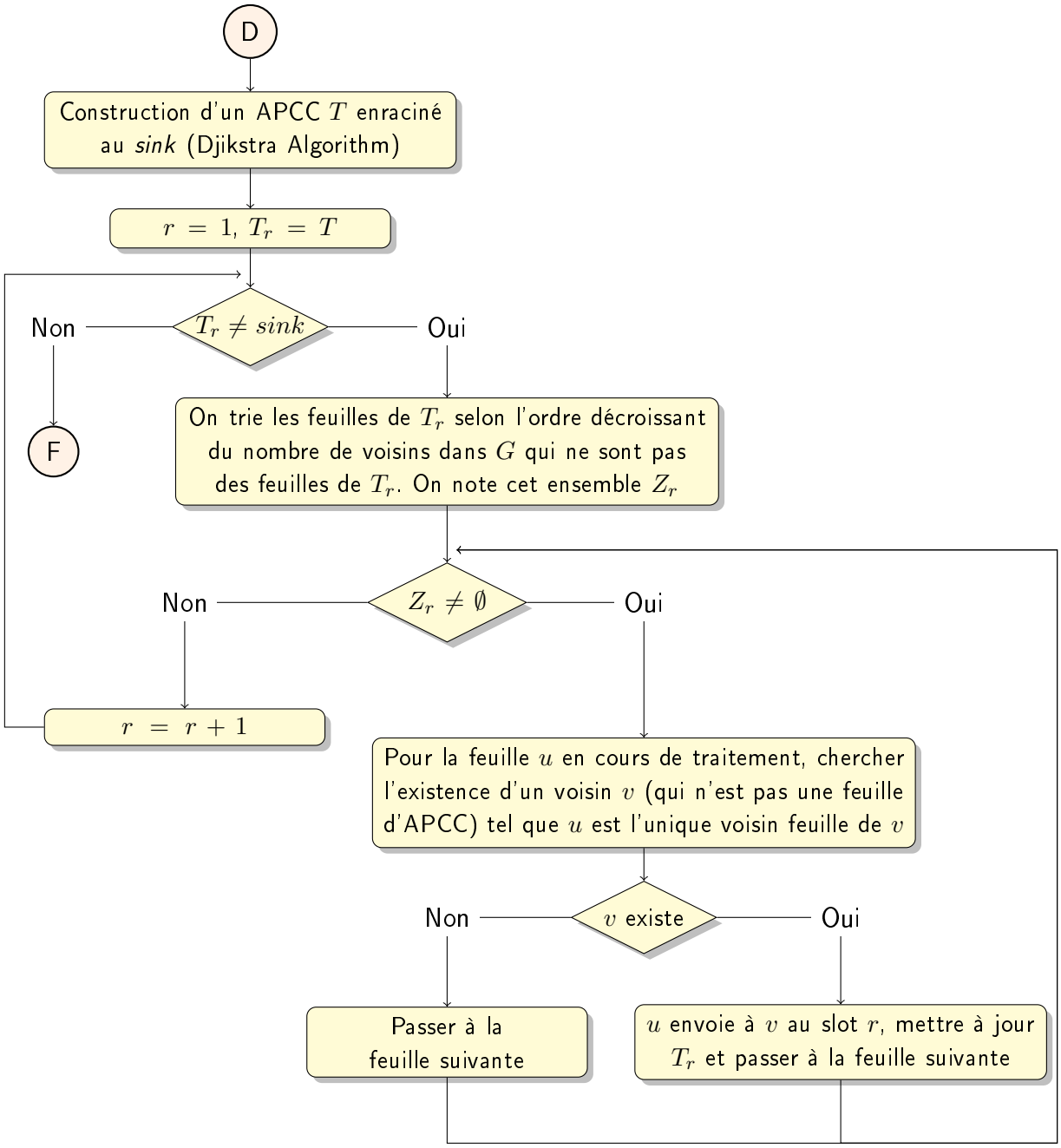


FIGURE 3.1 – Organigramme du protocole SDA

autre feuille comme voisin. Le *sink* a les sommets 5 et 6 comme voisins, donc 7 ne peut envoyer ses données au *sink*, tandis que le sommet 4 n'a que le 7 comme voisin. 7 choisit donc 4 comme parent au *slot* 1. L'arborescence est mise à jour en supprimant le sommet planifié.

On continue l'exécution de la même façon pour les autres sommets, si un sommet ne trouve pas de parent, il sera visité dans la prochaine itération.

Une itération se termine lorsque toutes les feuilles relevées auront été visitées. On passe ensuite au prochain *slot* et on réitère.

L'algorithme se termine lorsqu'il ne restera que le *sink* dans l'arborescence (plus aucun nœud à planifier).

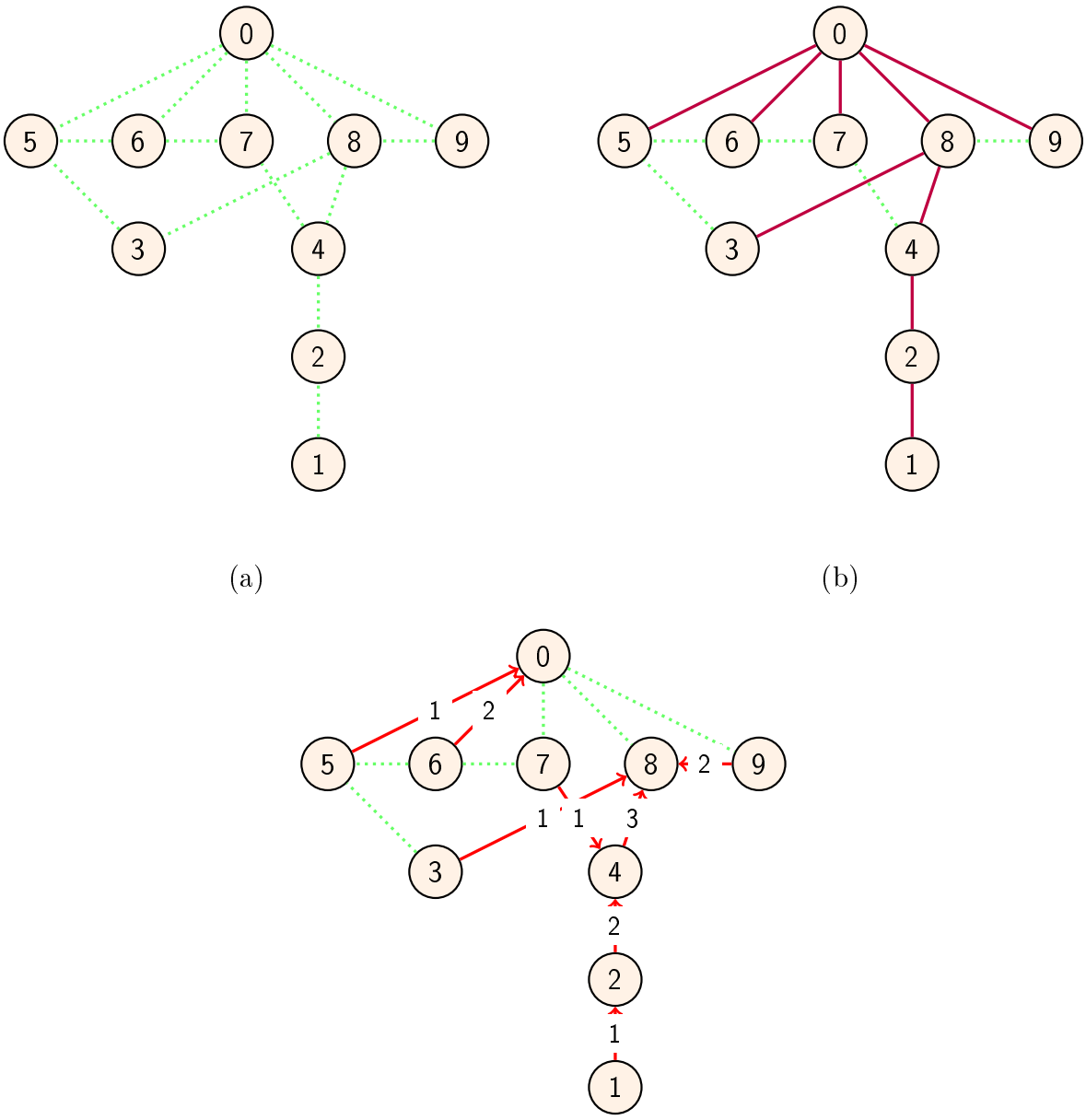


FIGURE 3.2 – Exemple d'exécution de la méthode SDA

3.2.2 Protocole NCA

Le protocole NCA (Nearly Constant Approximation) est apparu dans [Huang et al., 2007]. Huang et al. propose en 2007 une amélioration de la borne supérieure de la latence $(23R + \Delta - 18)$. Ce protocole est basé sur l'ensemble stable maximal, l'algorithme se déroule comme suit :

Avant d'aboutir à la planification de l'agrégation des données, NCA passe par trois phases. La première consiste à répartir le réseau en plusieurs niveaux et déterminer un ensemble stable maximal dans chacun d'eux. La deuxième phase construit l'arbre d'agrégation des données (en largeur d'abord) en connectant des nœuds niveau par niveau. Cet arbre est utilisé pour l'obtention d'un planning initial puis d'une planification finale. L'algorithme s'est révélé d'une grande efficacité sur les benchmarks.

Les failles de cette heuristique ont été découvertes en, dans [Yu et al., 2009] par Bo Yu et al. En effet, pour certaines instances, la planification résultante de l'exécution de NCA comporte des collisions. Le protocole a subi une correction et une autre borne supérieure $(24D + 6\Delta + 16)$ où D est le diamètre du réseau) a été proposée pour la latence.

3.2.3 Protocole ACS

Le protocole ACS (Aggregation Convergecast Scheduling) est apparu dans [Malhotra et al., 2011] avec une approche similaires aux autres. Les deux phases ont cependant été améliorées par Malhotra et al. en 2010.

3.2.4 Protocole DASUT

Dans [Bagaa et al., 2012], une nouvelle approche DSCA (Distributed Simultaneous tree Construction and data Aggregation scheduling) qui exécute simultanément la phase de la construction de l'arbre et celle de la planification a été proposée.

L'algorithme se déroule en deux phases : le pré-traitement du réseau, l'arbre d'agrégation et la planification simultanément.

- **Phase 1** : Pré-traitement

  On organise le graphe par niveaux (couches) :

  - Couche 1 : Les voisins du *centred sink*.
  - Couche 2 : Les voisins des voisins du *centred sink*. ⋮
  - Couche max : Les nœuds capteurs les plus éloignés du *centred sink*.

- **Phase 2** : Construction de l'arbre et planification

Nous allons définir les notations qui seront utilisées pour expliquer le déroulement de cette phase dans le tableau suivant :

Notation	Définitions
$\overline{TX}_u$	Les *slots* de transmission interdits pour u.
$\overline{RX}_u$	Les *slots* de réception interdits pour u.
$TChildMax_u$	Le *slot* durant lequel un fils de u peut transmettre ses données (pour assurer que u transmet après ses fils).
TS_u	*slot* visé par u pour envoyer ses données.
$\overline{SC}_u$	voisins de u qui ne sont pas encore planifiés.
$\eta(u)$	voisin de u dans le même niveau, et les niveaux juste en dessous et juste en dessus.
CPL_u	ensemble des noeuds candidats à être parents de u, $CS_u = n(u)$.
$Nodes_L$	ensemble de noeuds dans le niveau L.
$Ready\_List$	la liste des noeuds prêts, initialisés à $NULL$.
$Scheduled\_List$	liste des noeuds planifiés initialisés à $NULL$.
$Wait\_List$	liste des noeuds en attente.

TABLE 3.1 – Les notations utilisées dans DASUT [Bagaa et al., 2012]

Étapes de l'algorithme DASUT

I. On commence par initialiser $Ready\_List$ à $Nodes_L$ $Scheduled\_List$.

II. Pour chaque nœud u dans $Ready\_List$ déterminer TS_u de sorte que : $TS_u > TChildMax_u$ et $TS_u \notin \overline{TX}_u$ et $TS_u \notin \overline{RX}_u$

III. Pour chaque nœud u déterminer l'ensemble des noeuds candidats à être parents de u de façon à respecter les conditions suivantes :

   (i) Si w est un nœud voisin de u faisant partie du niveau $L-1$ alors w est ajouté à CPL_u pour le *slot* TS_u.

   (ii) Si w est un nœud voisin de u faisant partie du niveau L alors w est ajouté à CPL_u si et seulement si :
   $\exists t\ TS_u \leqslant t \leqslant TS_w$ et $t \notin \overline{TX}_u$ et $t \in \overline{RX}_u$

   (iii) Si w est un nœud voisin de u faisant partie du niveau $L+1$ alors w est ajouté à CPL_u si et seulement si :
   $\exists t\ TS_u \leqslant t < TS_w$ et $t \notin \overline{TX}_u$ et $t \in \overline{RX}_u$

IV. Pour chaque nœud u déterminer le meilleur parent (le candidat à être parent qui a le moins de voisins non planifiés).

V. Soit w le meilleur parent de u , pour chaque u on calcule : $C(u,w) = (\overline{SC}_u, \overline{SC}_w, TS_u)$
   Pour éviter les collisions et les chevauchements, on vérifie :

   (i) L'existence de $v \in Ready\_List$ tel que v a z comme parent.

(ii) Si $C(v,z) < c(u,w)$ et ($w \in \eta(u)$ ou $z \in \eta(u)$) alors : on place u dans $Wait\_List$.

(iii) Si non, u est planifié au *slot* TS_u

- On met à jour $\overline{RX}_v$ ($\overline{RX}_v = \overline{RX}_v \ \{TS_u\}$)
- On met à jour $\overline{TX}_v$ ($\overline{TX}_v = \overline{TX}_v \ \{TS_u\}$)
- $TChildMax_u = Max(TChildMax_w, TS_u)$

VI. Si $Ready\_List$ est vide, on effectue le test suivant :

(i) Si tous les nœuds du niveau actuel L ont été planifiés alors on passe au niveau suivant et on recommence de (I).

(ii) Si non on pose $Ready\_List = Wait\_List$ et on recommence de (II).

3.2.5 Protocole DASUT et SDA amélioré

Dans [Yagouni et al., 2015], un protocole regroupant les avantages des heuristiques citées précédemment en évitant leurs limites, a été présenté par Yagouni et al. en 2015.

Afin d'optimiser la planification, il a été proposé de modifier l'arbre d'entrée du protocole SDA. La première étape consiste à appliquer le protocole DAS-UT afin d'obtenir une arborescence initiale qui sera utilisée pour la seconde étape.

Les modifications apportées au protocole SDA sont les suivantes :

- Les feuilles seront examinés dans l'ordre croissant du nombre de voisins qui ne pas des feuilles.
- Les éléments de l'ensemble S qui ne seront pas planifiés à une étape donnée sont placés dans l'ensemble des sommets qui ne sont pas des feuilles $\overline{S}$ afin de donner plus de possibilités dans le choix du parent pour les éléments de l'ensemble S.

Étapes de l'algorithme hybridation DASUT et SDA amélioré

Dans cette méthode, la construction de l'arbre d'agrégation et la planification s'exécutent simultanément. L'arbre est créé de la méthode DASUT, ensuite il est utilisé comme entrée pour l'algorithme de planification. Nous soulignons l'arbre obtenu ne sera pas considéré comme un arbre d'agrégation de données, il utilisé pour trier les nœuds du réseau dans le processus de planification.

Ce processus est exécuté en plusieurs itérations, on détermine d'abord l'ensemble de nœuds qui peuvent être planifiés au même *slot*. Les feuilles de l'arbre de DASUT sont triées selon l'ordre croissant du nombre de voisins qui ne sont pas des feuilles.

Soient S et $\overline{S}$, les ensembles contenant les feuilles triées et les nœuds qui ne sont pas des feuilles de l'arbre obtenu de DASUT, respectivement. Pour tout nœud u dans S, le test suivant sera effectué : Si $\exists\ v\ \in \overline{S}$ où v est le voisin privé de u (i.e. $\nexists$x$\in S - \{u\}$, dans lequel w est un voisin de v), alors u est planifié et v est sélectionnée comme parent dans ce *slot*. Ici, u est retiré de l'arbre de DASUT. Sinon, v sera intégré dans $DAS-UT$

Lorsque tous les nœuds dans S sont testés, les nœuds feuilles qui ne sont pas planifiés ainsi que les nouveaux nœuds feuilles (dans l'arbre de DASUT) sont considérés pour la prochaine itération, et seront triés et testés en utilisant la même approche.

Exemple de l'exécution de SDA amélioré

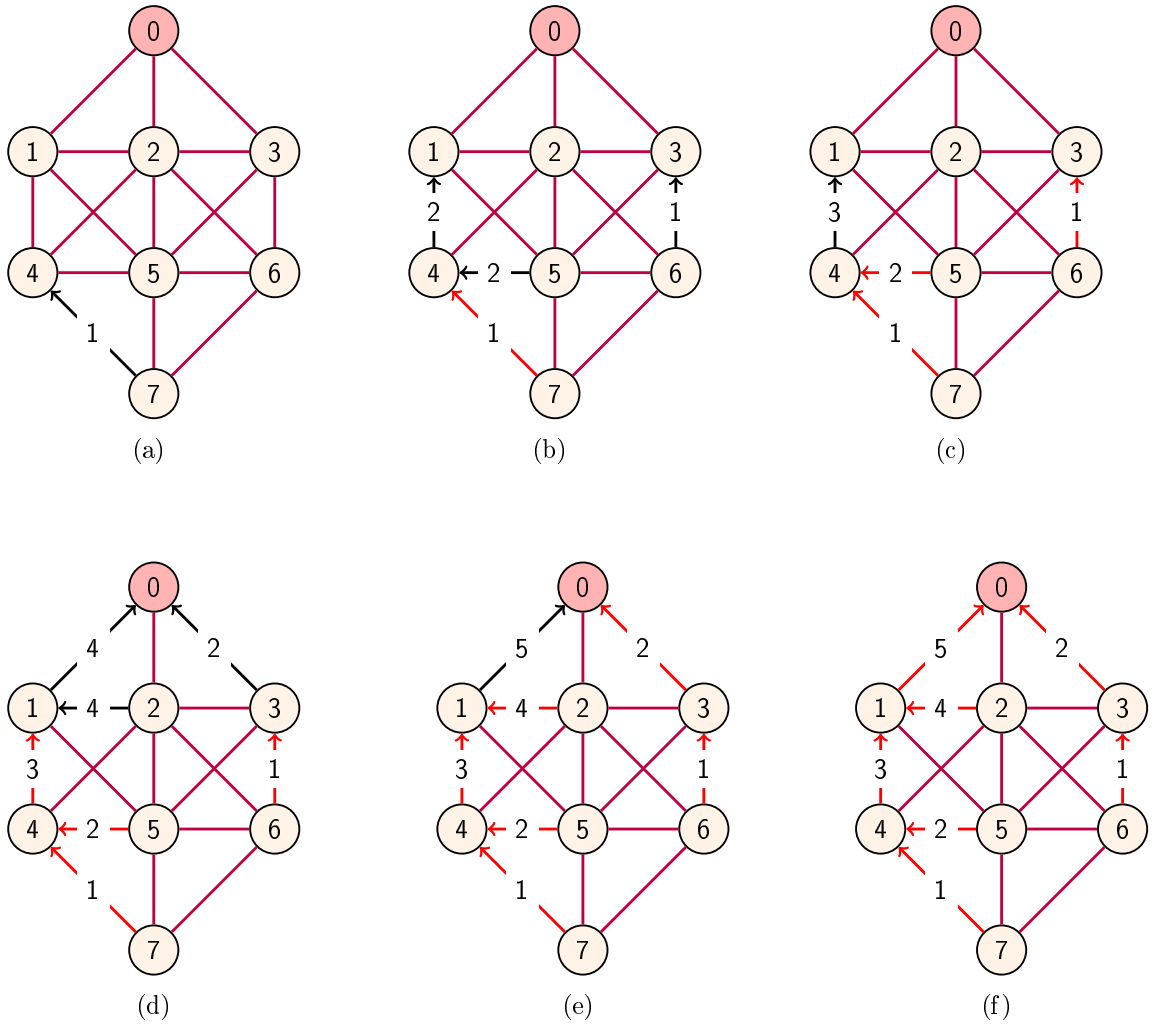


FIGURE 3.3 – Construction de l'arbre de DAS-UT

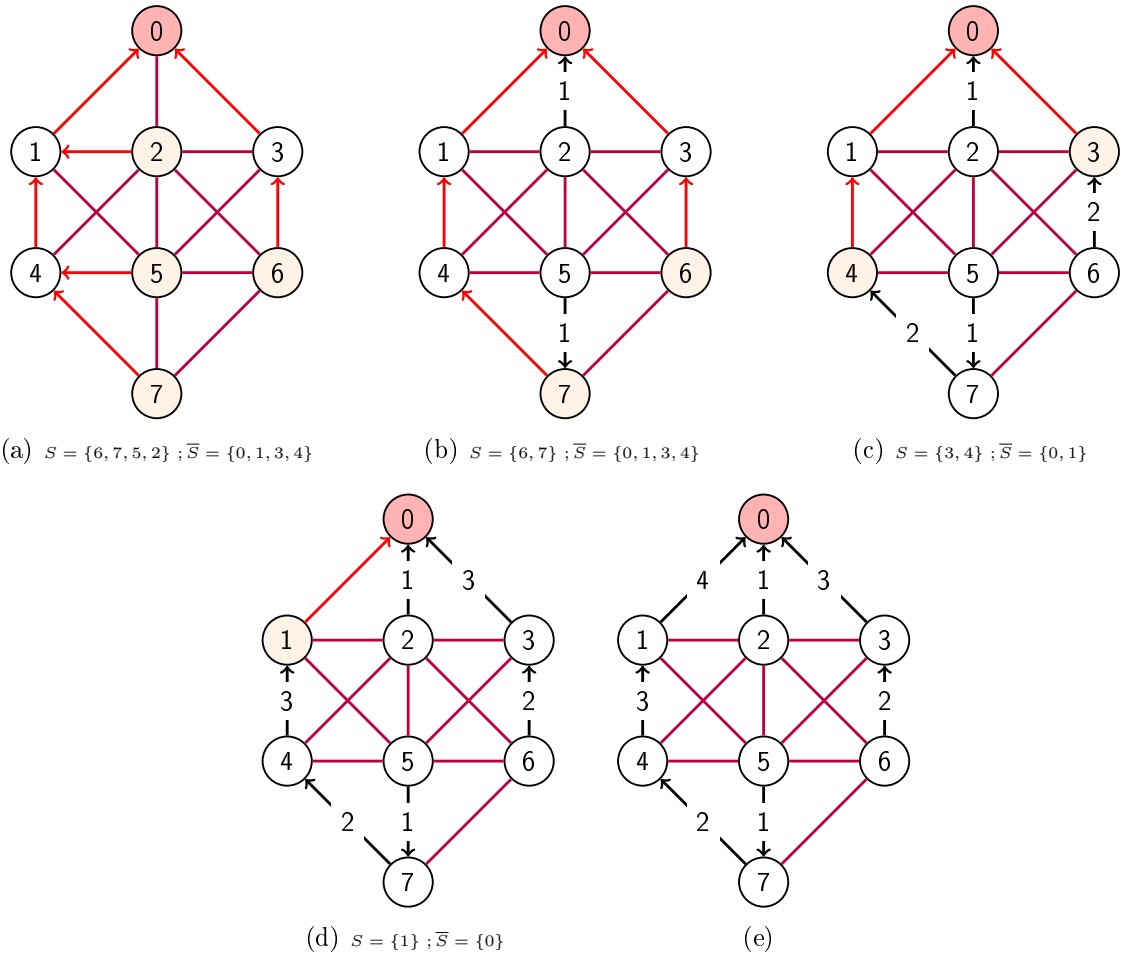
Dans la figure 3.3, les étapes de la construction de l'arbre par le protocole DASUT sont illustrées.

Dans (a), le noeud 7 choisit le parent ayant le moins de voisins non planifiés, ce choix est validé dans (b). On passe ensuite au niveau suivant qui contient les nœuds 4, 5 et 6.

A cette étape, le noeud 4 choisit le nœud 1 comme parent, il pourra donc lui transmettre ses données *slot* 2, et le nœud 6 a choisi le noeud 3 pour le *1*.

Trois triplets seront construits pour le noeud 4 : $(2, 3, 3)$, pour le noeud 5 : $(2, 3, 5)$ et pour le noeud 6 : $(1, 3, 3)$, ce qui qualifie les noeuds 4 et 6 à être validés pour cette itération et le noeud 5 va changer son parent et son *slot* pour la prochaine itération et ainsi de suite jusqu'à la fin de la construction de l'arbre (f).

Dans la figure précédente, la méthode SDA-modifié a été appliquée pour la construction et la planification de l'arbre d'agrégation. Comme le montre (a), la première étape est de construire et trier l'ensemble S contenant les feuilles de l'arbre de DAS-UT : $\{6, 7, 5, 2\}$, puis l'ensemble $\overline{S}$ qui contient les noeuds non-feuilles : $\{0, 1, 3, 4\}$. Le noeud 6 n'a pas de voisin privé dans $\overline{S}$, il sera donc retiré de S et intégré dans $\overline{S}$, et de même pour le noeud 7. Le noeud 5 a un voisin privé, et c'est le noeud 7 qui a été intégré dans cette itération dans $\overline{S}$. Le noeud 2 a un voisin privé dans $\overline{S}$ qui est le noeud 0. Donc, les



(a) $S = \{6, 7, 5, 2\}$; $\overline{S} = \{0, 1, 3, 4\}$ (b) $S = \{6, 7\}$; $\overline{S} = \{0, 1, 3, 4\}$ (c) $S = \{3, 4\}$; $\overline{S} = \{0, 1\}$

(d) $S = \{1\}$; $\overline{S} = \{0\}$ (e)

FIGURE 3.4 – Planification par SDA-modifié

noeuds 5 et 2 prennent comme parents les noeuds 0 et 7 respectivement dans le *slot* 1. Dans (b),les ensembles S et $\overline{S}$ sont mis à jour, et le *slot* 1 devient 2, et la même façon, les noeuds restants sont planifiés jusqu'à l'obtention de l'arbre d'agrégation planifié (e).

3.3 Modèles mathématiques et agrégation des données

3.3.1 Première modélisation mathématique

Dans [Tian et al., 2011] un modèle mathématique a été donné par Tian et al. permettant l'obtention de la solution optimale au problème pour des instances de petite taille (10 capteurs et un nœud puits).

Ils ont défini, à partir du graphe $G = (V, E)$ représentant un RCSF à N sommets, un graphe auxiliaire noté $\overrightarrow{G'} = (V', \overrightarrow{E'})$ tel que :

- La construction du graphe $\overrightarrow{G}$ est faite de façon à ce que si une arête existe dans G entre deux sommets i et j, elle est remplacée dans $\overrightarrow{G}$ par deux arcs : de i vers j et de j vers i.

FIGURE 3.5 – Construction des arcs du graphe auxiliaire

- Les arcs sortant du *sink* dans $\overrightarrow{G}$ sont enlevés (le *sink* n'envoie pas de données).
- Un sommet jouant le rôle d'une station de base fictive est ajouté.
- Un arc du *sink* vers le *sink* fictif est également ajouté.

N	Le nombre de nœuds dans $\overrightarrow{G}$; il y a $N+1$ noeuds dans $\overrightarrow{G'}$.
i	Un nœud dans $\overrightarrow{G'}$.
M	Le nombre d'arcs dans $\overrightarrow{G'}$.
e	Un arc dans $\overrightarrow{G'}$.
$e_{K'}$	La dernière arête dans $\overrightarrow{G'}$.
t	*slots* allant de 1 à N.
$S(i)$	L'ensemble de toutes les arêtes avec i comme émetteur dans $\overrightarrow{G'}$.
$D(i)$	L'ensemble de toutes les arêtes avec i comme récepteur dans $\overrightarrow{G'}$.
$W(e)$	L'ensemble des arêtes dans $\overrightarrow{G'}$, contenant les arêtes susceptibles d'avoir des collisions avec le récepteur de e et les arêtes pouvant être affectées par la transmission de e.
$x_{e,t}$	Les variables de décision. $x_{e,t} = 1$ si l'arête e est programmée pour transmettre durant le *time slot* t ; et 0 si non.

TABLE 3.2 – Les notations utilisées

Une solution évidente pour $\overrightarrow{G'}$ avec $K' = N$ est que tous les nœuds, sauf b', envoient des leurs données (un par un) le long d'un arbre. Donc, K' est forcément plus petit que N.

$$\begin{cases} Min\ K' = 1 * x_{e_{K'},1} + 2 * x_{e_{K'},2} + ... + N * x_{e_{K'},N} \\ \sum_{e \in S(i)} \sum_{t=1}^{N} x_{e,t} = 1 \quad \forall i \in \overrightarrow{G'} \setminus b' & \text{(a)} \\ \sum_{t'=t+1}^{N} x_{e',t'} \leq 1 - \sum_{e \in S(i)} x_{e,t}, \\ \forall i \in \overrightarrow{G'}\ \forall e' \in D(i)\ \forall t & \text{(b)} \\ x_{e',t} \leq 1 - x_{e,t}\ \forall e\ \forall e' \in W(e)\ \forall t & \text{(c)} \\ \sum_{e \in S(i)} x_{e,t} + \sum_{e' \in D(i)} x_{e',t} \leq 1\ \forall i \in \overrightarrow{G'}\ \forall t & \text{(d)} \end{cases}$$

Le groupe de contraintes (a) assure que la contrainte de transmission est respectée pour chaque nœud. Avec $t' = t+1, ..., N$, le groupe de contraintes (b) vérifie que chaque

nœud ayant transmis ses données, ne peut être récepteur d'aucune autre transmission. Le groupe de contraintes (c) permet d'obtenir une solution qui ne contient pas d'interférences. Le groupe de contraintes (d) assure qu'un nœud émetteur ne peut recevoir en même temps qu'il transmet.

3.3.2 Amélioration du modèle mathématique

Dans [Yagouni et al., 2015], un modèle mathématique a été donné, permettant l'obtention de la solution optimale au problème pour 30 capteurs et un nœud puits.

On considère un réseau de N capteurs et un nœud puits.

On note E l'ensemble des arêtes et $V(i)$ l'ensemble des voisins du sommet i.

Les variables de décision sont définies comme suit :

- T est la variable qui désigne la latence.
- $\forall (i,j) \in E$ et $\forall t = 1, ..., N$ les variables sont définies par :

$$x_{(i,j),t} = \begin{cases} 1 & \text{si l'arête } (i,j) \text{ prend la période t} \\ 0 & \text{sinon} \end{cases}$$

Le modèle s'écrit :

$$\begin{cases} Min \;\; T & (0) \\ t \times x_{(i,j),t} \leq T \;\; \forall (i,j) \in E \;\; \forall t = 1, \ldots, N & (1) \\ \sum\limits_{j \in V(i)} \sum\limits_{t=1}^{N} x_{(i,j),t} = 1 \;\; \forall i = 1, \ldots, N & (2) \\ x_{(i,j),t} + x_{(j,k),t'} \leq 1 \\ \forall (i,j) \in E \;\; \forall (j,k) \in E \;\; \forall t, t' = 1, \ldots, N \mid t' \leq t & (3) \\ \sum\limits_{i \in V(j)} x_{(i,j),t} \leq 1 \;\; \forall j = 0, \ldots, N \;\; \forall t = 1, \ldots, N & (4) \\ x_{(i,j),t} + x_{(k,l),t} \leq 1 \\ \forall (j,k) \in E \; \forall i \in V(j) \;\; i \neq k \; \forall l \in V(k) \; j \neq l \\ \forall t = 1, \ldots, N & (5) \\ x_{(i,j),t} \in \{0,1\} \; \forall (i,j) \in E \; \forall t = 1, \ldots, N \end{cases}$$

- La fonction objectif consiste en la minimisation de la latence.
- Le premier groupe de contraintes exprime que quel que soit le *slot* choisi pour une arête (i,j), ce *slot* doit être inférieur ou égal à la latence T.
- Le deuxième groupe de contraintes exprime que chaque nœud transmet exactement une fois.
- Le troisième groupe désigne les contraintes de précédence.
- Le quatrième groupe désigne les contraintes de chevauchement.
- Le cinquième groupe désigne les contraintes de collision.

3.4 Classification du problème

Le problème qui consiste à minimiser la latence lors de l'agrégation des données dans un réseau de capteurs sans fil est classé parmi les problèmes NP-difficile. En effet, Chen et al. ont établi la preuve de la NP-dureté dans [Chen et al., 2005].

La preuve se base sur une réduction polynomiale effectuée à partir d'une variante du problème 3-SAT.

4 Modèle proposé et méthodes de résolution

4.1 Introduction

Lorsqu'on est face à un problème NP-difficile, la modélisation par des outils mathématiques est l'un des moyens utilisés permettant d'obtenir une solution exacte (donc optimale). Pour ce problème, nous avons pensé à changer notre vision et le modéliser, en le ramenant à un problème de satisfaction de contraintes que nous définirons.

4.2 Modèle proposé

Dans cette section, [Apt, 2003] a été utilisé comme référence principale.

4.2.1 Problème de satisfaction de contraintes

Un problème de satisfaction des contraintes (*Constraint Satisfaction Problem*, CSP) est un problème modélisé sous la forme d'un ensemble de contraintes posées sur des variables, chacune de ces variables prend des valeurs dans un domaine. De façon plus formelle, un CSP est défini par un triplet (X, D, C) tel que :

- $X = \{X_1, X_2, ..., X_n\}$ est l'ensemble des variables (les inconnues) du problème
- D est la fonction qui associe à chaque variable X_i son domaine $D(X_i)$
- $C = \{C_1, C_2, ..., C_k\}$ est l'ensemble des contraintes, chaque contrainte C_j est une relation entre certaines variables de X, restreignant les valeurs que peuvent prendre simultanément ces variables.

Une contrainte $C_{i,j,k,..}$ entre les variables $X_i, X_j, X_k, ...$ est une relation définie sur les domaines correspondants aux différentes variables.

La contrainte est un sous-ensemble de combinaisons possibles des valeurs que peuvent prendre les variables [1]. Une solution au CSP serait donc d'affecter à chacune des variables, une valeur de son domaine de sorte que toutes les contraintes soient satisfaites.

Un CSP qui admet une solution est dit consistent, sinon il est dit inconsistant.

1. D'une manière équivalente, une clause dans le problème SAT

4.2.2 Problème d'optimisation avec contraintes

Lorsqu'un problème d'optimisation (minimisation ou maximisation) est modélisé par un CSP, en lui ajoutant une fonction objectif,

De nombreuses techniques très puissantes pour résoudre ce genre de problèmes (CSP et COP) ont été proposées, plusieurs d'entre elles utilisent les métaheuristiques.

4.2.3 Modélisation CSP du problème

Cette modélisation assure que les contraintes liées au problème sont respectées. Elle permet de donner une solution réalisable. Nous verrons son intérêt quelques sections plus loin dans ce chapitre.

En gardant les mêmes notations que dans le modèle précédent, on définit l'ensemble des variables comme suit :

$$X = \{x_{st}, \quad s \in \{1, \ldots, N\}\}, \quad t \in \{1, \ldots, N\}\}$$

$$x_{st} = \begin{cases} -1 & \text{si le noeud s n'effectue aucune transmission au slot } t \\ 0 & \text{si le noeud s transmet au sink au slot } t \\ 1, \ldots, N & \text{pour les transmissions internodales au slot } t \end{cases}$$

- Un ensemble D de domaines :
  x_{st} prend ses valeurs dans $V(s)$ (voisin de s).
- Un ensemble C de contraintes :
  - *Contraintes de transmissibilité* : Chaque nœud transmet exactement une fois.

$$\forall s = 1, ..., N \quad |\{x_{st} \neq -1 | t = 1, ..., N\}| = 1$$

  - *Contraintes de précédence* : Si le nœud s transmet au nœud r au *slot* t et le nœud r transmet au *slot* t' alors on doit avoir $t < t'$

$$\forall r = 1, ..., N \; \forall s \in V(r) \;\; \forall t, \; t' = 1, ..., N \; x_{st} = r \; et \; x_{rt'} \neq -1 \Rightarrow \;\; t < t'$$

  - *Contraintes de chevauchement* : Si le nœud s transmet à r au *slot* t alors le nœud r ne peut recevoir une autre transmission au même *slot*.

$$\forall s = 1, ..., N \; s' \in V(r) \; \forall t = 1, .., N \;\; x_{st} = r \Rightarrow x_{s't} \neq r$$

  - *Contraintes de collision* : Si les nœuds j et k sont voisins et k transmet au*slot* t alors j ne peut recevoir une transmission au même *slot*.

$$\forall k = 1, ..., N \; \forall j \in V(k) \; \forall i \in V(j) \; \forall t = 1, .., N \;\; : x_{kt} \neq -1 \Rightarrow x_{it} \neq j$$

4.2.4 Modélisation par un problème d'optimisation contraint

La formulation COP [2] du problème de minimisation de la latence lors du processus d'agrégation des données dans les RCSF utilise les ensembles de variables, de domaines et de contraintes que le CSP. On ajoute cependant la fonction à minimiser qui est exprimée comme suit :

$$\begin{cases} Min \quad Max_{t=1,...,N}\{t(x_{st}+1)|\forall s \in V(0)\} \\ \text{Sous les contraintes :} \\ \forall s = 1,...,N \quad |\{x_{st} \neq -1 | t = 1,...,N\}| = 1 \\ \forall r = 1,...,N \; \forall s \in V(r) \;\; \forall t,\; t' = 1,...,N \; x_{st} = r \; et \; x_{rt'} \neq -1 \Rightarrow \;\; t < t' \\ \forall s = 1,...,N \; s' \in V(s) \; \forall t = 1,..,N \;\; x_{st} = r \Rightarrow x_{s't} \neq r \\ \forall k = 1,...,N \; j \in V(k) \; i \in V(j) \; \forall t = 1,..,N \;\; : x_{kt} \neq -1 \Rightarrow x_{it} \neq j \end{cases}$$

En minimisant la latence, on cherche à minimiser le *slot* maximum, auquel envoie les voisins du *sink*.

Complexité du modèle

- *Nombre de contraintes* : $N \times N \times N + 3 \times N \times N$ donc $O(N^3)$
- *Nombre de variables* : $N \times N$ donc $O(N^2)$

4.3 Méthodes de résolution (Heuristiques)

Le modèle mathématique présenté est formé de contraintes logiques. Les solvers d'optimisation permettant de résoudre ce type de problème (non linéaire) sont nombreux. Cependant, nous n'avons accès qu'aux versions d'essais de ces logiciels, pouvant contenir au maximum 300 contraintes et 300 variables.

De ce fait, nous nous penchons sur les méthodes approchées de type heuristiques pour résoudre le problème.

4.3.1 Heuristique proposée (Visited = Planified)

Cette heuristique s'inspire de l'étude de synthèse réalisée suite à la préparation de l'état de l'art. Elle se base essentiellement sur les remarques et les analyses qui nous ont permis d'identifier les points forts et les limites des protocoles étudiés et des méthodes existantes :

A l'exemple de la méthode DASUT, on commence par faire un pré-traitement sur le graphe (ordonner les sommets niveau par niveau). En commençant du niveau le plus profond (le plus bas), on ordonne les sommets du niveau actuel par ordre croissant des degrés (en ne prenant en considération que les voisins non encore planifiés).

2. COP : Contrained Optimization Problem

En parcourant les sommets du niveau courant dans l'ordre déterminé. Pour chaque noeud u on va :

- Poser $slot_u = 1$
- Parcourir les voisins un par un (en commençant par les voisins du niveau supérieur puis les voisins du même niveau que u ensuite les voisins du niveau inférieur) et vérifier si pour un voisin w les contraintes de précédence, de chevauchement et de collision sont respectées. Si tel est le cas, alors u prend w comme parent au slot actuel.
- Si aucun voisin ne peut être parent de u alors on passe au slot suivant et on réitère.

L'algorithme se termine lorsque tous les sommets auront été visités.

L'organigramme de la méthode "Visited = Planified" est présenté dans la figure 4.1.

Dans la Figure 4.2, l'exécution de la méthode commence au niveau 4, qui ne contient que le nœud 1, pour $slot = 1$, le sommet peut transmettre à son unique voisin 2.

Après avoir planifié tous les noeuds du niveau 4, la transition est faite au niveau 3 qui ne contient que le sommet 2. Pour $slot = 1$ une contrainte de précédence n'est pas respectée, on passe alors au *slot* suivant ($slot = 2$), le sommet peut transmettre à son unique voisin 4.

Après avoir planifié tous les sommets du niveau 3, on passe au niveau 2 qui contient les noeuds 3 et 4. Les deux sommets ont deux voisins non encore planifiés, on choisit arbitrairement l'un d'eux.

Supposons qu'on ait choisit le sommet 3, pour $slot = 1$, ce sommet peut transmettre à 5 ou à 8. On choisit arbitrairement l'un des deux voisins (le sommet 5 par exemple).

Il ne reste alors que le sommet 4 qui n'a pas encore été planifié au niveau 2. Pour $slot = 1$ et 2. La contrainte de précédence n'est pas respectée. Suite à l'incrémentation du *slot* (3), le sommet a la possibilité d'envoyer aux voisins 7 ou 8. Il choisit arbitrairement le voisin 8.

On passe alors au dernier niveau qui contient les sommets 5, 6, 7, 8 et 9. Ils seront visités dans l'ordre suivant : 8, 9, 5, 7, 6.

On commence par le noeud 8, pour $slot = 1$, 2 et 3, la contrainte de précédence n'est pas respectée. On planifie 8 pour qu'il transmette au *sink* au *slot* 4.

Le noeud 9 peut envoyer au *sink* au *slot* 1.

Le sommet 5 peut envoyer au *sink* ou à 6 au *slot* 2. Il privilégie le *sink* (car il se trouve au niveau supérieur).

De la même manière, le sommet 7 est planifié au $slot = 3$ pour transmettre au *sink* (pour les *slots* 1 et 2, les contraintes de précédence et de collision ne sont pas respectées).

Le sommet 6 ne peut envoyer au *slot* 1 (un chevauchement et une collision sont créés). Il envoie dont au sommet 7 au *slot* 2.

Nous obtenons la même valeur de la latence en appliquant SDA et l'heuristique proposée. Nous remarquons cependant que l'arbre d'agrégation obtenu n'est pas le même pour les deux méthodes.

Ce qui constitue un plus dans la méthodologie de traitement et la suite que nous envisageons, à savoir, l'initialisation de méthodes plus sophistiquées à l'image des métaheuristiques.

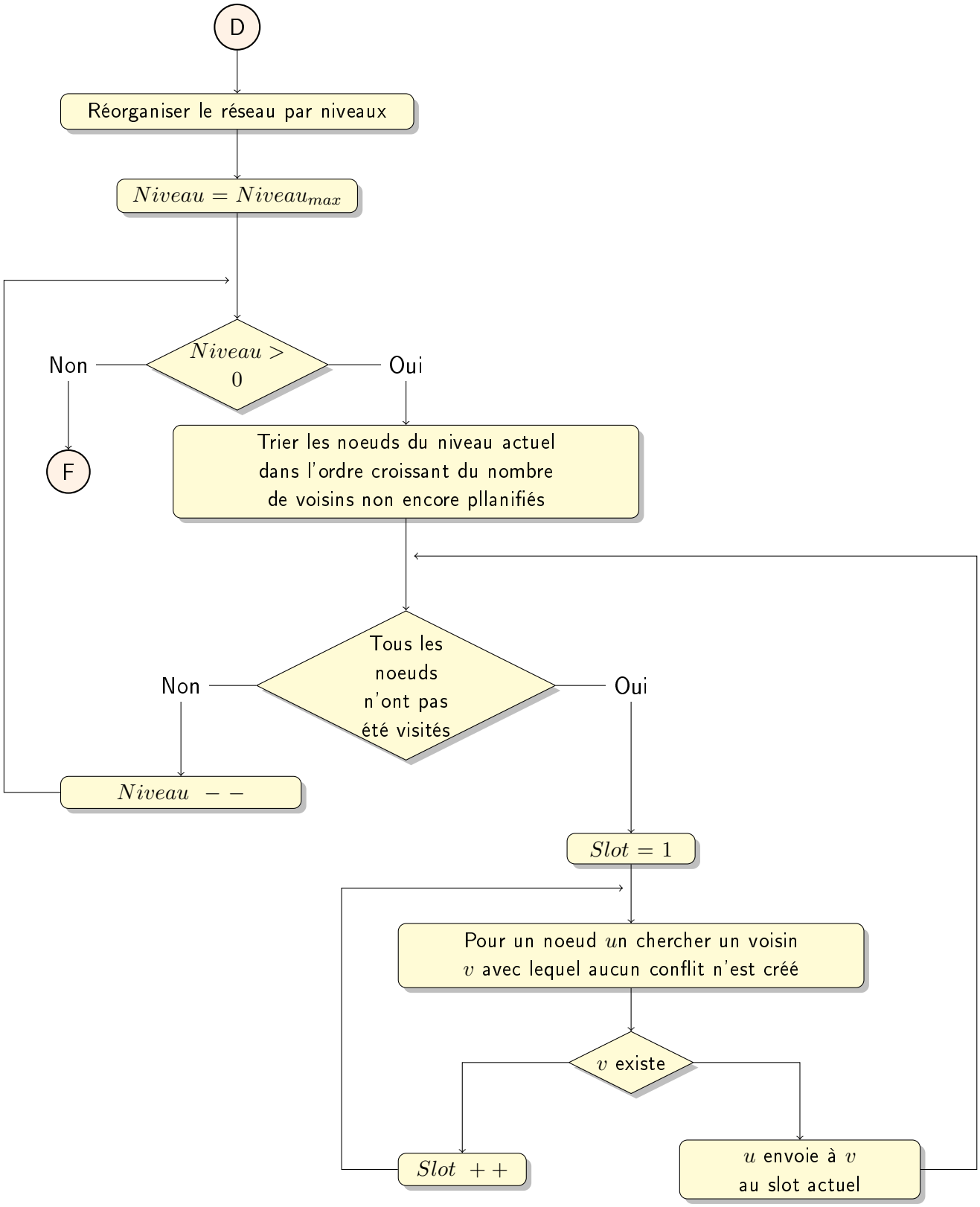
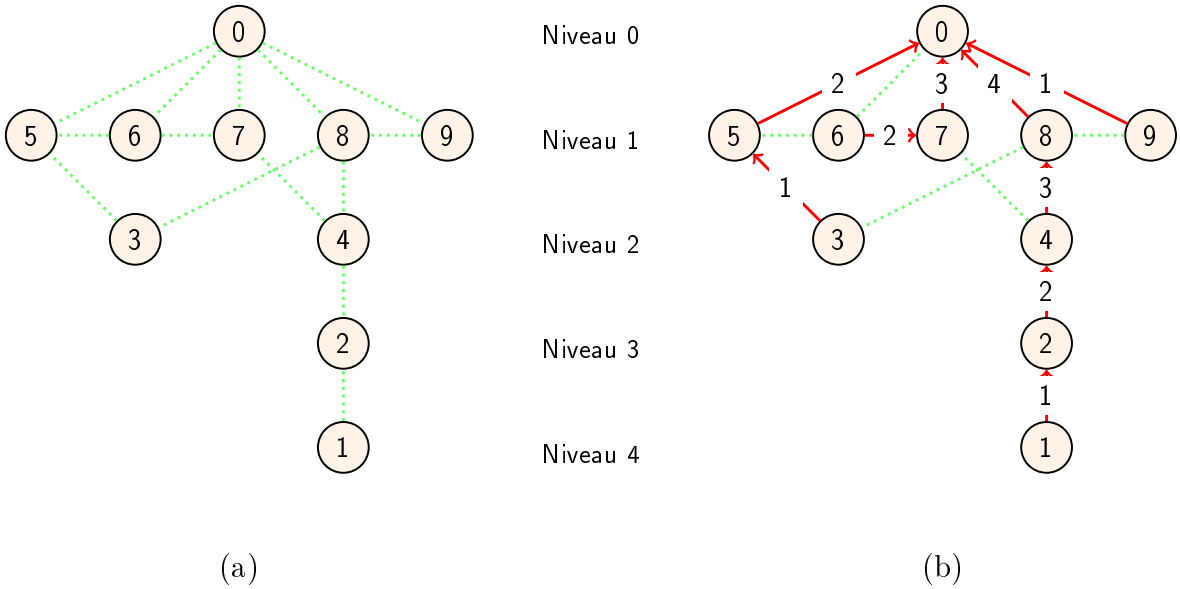


FIGURE 4.1 – Organigramme de la méthode "Visited = Planified"



(a) (b)

FIGURE 4.2 – Exemple de l'exécution de "Visited = Planified"

4.3.2 Heuristique basée sur le CSP

Dans cette heuristique, nous commençons par faire un pré-traitement sur le réseau, pour l'organiser par niveaux (comme dans DASUT). Nous notons à cet effet, $MaxLayer$ le niveau le plus bas et $ActualLayer$ le niveau en cours d'exécution.

Pour chaque niveau, nous déterminons l'ensemble de ses nœuds. Chaque nœud choisira un voisin, aléatoirement, du niveau supérieur pour lui transmettre ses données (parent) au *slot* défini par $MaxLayer - ActualLayer + 1$. Nous réitérons cette procédure, jusqu'à "planifier" tous les nœuds, sauf ceux qui sont directement liés au *sink*, nous effectuerons leur traitement dans la dernière étape de cet algorithme.

La solution ainsi construite ne sera pas réalisable. En effet, si deux nœuds d'un même niveau choisissent le même parent, et envoient au même *slot*, cela créera alors un chevauchement. Nous pouvons aussi rencontrer des cas de collision.

Quant aux contraintes de précédence, dans la solution construite, elles sont toutes respectées initialement (par construction).

Dans l'élaboration de la solution initiale, chaque nœud envoie exactement une seule fois. Les contraintes de transmissibilité sont de ce fait, toutes respectées.

Dans la suite de cette méthode, nous allons "corriger" la solution initiale, niveau par niveau, en commençant du niveau le plus bas (les nœuds les plus éloignés du *sink*), dans le but d'obtenir une solution réalisable. Pour ce faire, cette procédure de "réparation" se déroule comme suit :

(i) Déterminer pour chaque nœud du niveau courant, le nombre de contraintes non respectées engendrées par sa planification.

(ii) Trier les nœuds selon l'ordre décroissant du nombre de contraintes qui ne sont pas respectées.

(iii) Visiter les nœuds triés en faisant en sorte d'augmenter le *slot*, lorsque cela est nécessaire, pour que les contraintes de précédence avec les voisins du niveau inférieur soient respectées.

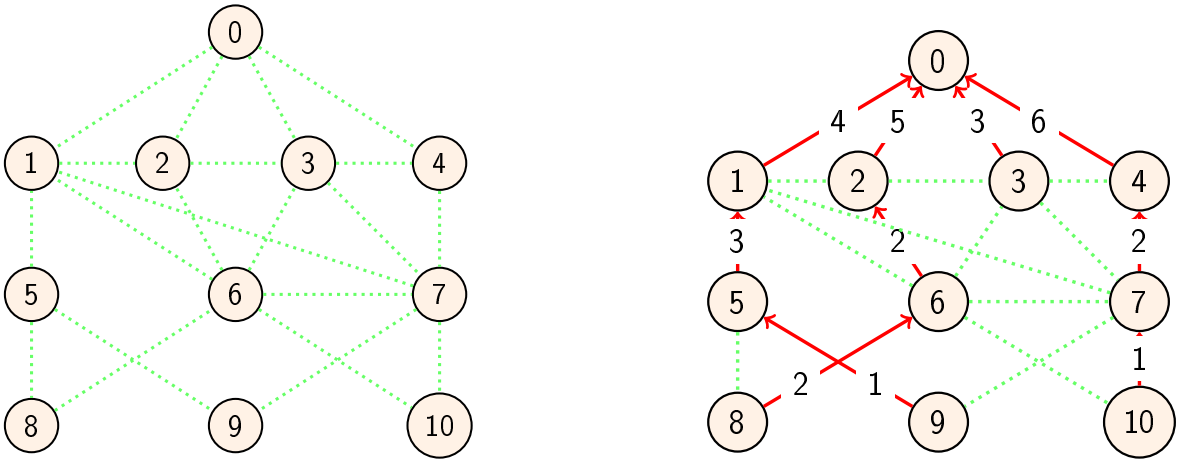
(iv) Visiter les nœuds triés en faisant en sorte de respecter les contraintes de collision.

(v) Visiter les nœuds triés en faisant en sorte de respecter les contraintes de chevauchement.

Pour chaque sommet u du niveau 1 : nous appliquerons le principe de l'heuristique "**Visited = Planified**" comme suit :

- Poser $slot = 1$
- Parcourir les voisins un par un (en commençant par le *sink* puis les voisins du même niveau que u) et vérifier si pour un voisin w les contraintes de précédence, de chevauchement et de collision sont respectées. Si tel est le cas, alors u prend w comme parent au slot actuel.
- Si aucun voisin ne peut être parent de u alors on passe au slot suivant et on réitère.

L'algorithme prend fin lorsque le nombre de contraintes non respectées est réduit à 0.



(a) (b)

FIGURE 4.3 – Exemple d'exécution de la méthode basée sur le CSP

Ces deux heuristiques pourront être utilisées comme moyen de diversification dans les voisinages visitées par les métaheuristiques. Après les avoir appliqué sur plusieurs exemples, nous avons constaté que les arbres d'agrégation résultants sont différents.

4.4 Métaheuristiques et agrégation des données

Dans cette section, nous allons expliquer l'adaptation de deux métaheuristiques sur le problème de la minimisation de la latence, lors du processus d'agrégation des données dans les réseaux de capteurs sans fil, que nous avons empruntée du travail réalisé par Djaoui et Mobasti dans la cadre de leur mémoire de Master, portant sur la contribution à l'optimisation de la planification de la collecte de données agrées dans un réseau de capteurs sans fil.

Nous avons choisi le recuit simulé et la recherche à voisinages variables. Leurs principes ont été expliqués dans le premier chapitre.

4.4.1 Recuit Simulé (RS)

Pour l'adaptation du recuit simulé, nous aurons besoin de définir une transformation admissible qui sera utilisée comme voisinage et de choisir une solution initiale pour démarrer l'algorithme.

Transformation Admissible

Nous avons opté pour une transformation admissible simple qui consiste à déterminer le meilleur parent dans le graphe pour un sommet i choisi aléatoirement (changer le parent de i pour l'affecter à un *slot* inférieur).

Solution initiale

Pour que la solution de départ soit réalisable, il est possible de prendre une solution résultante d'une heuristique. Nous pouvons par exemple prendre la solution obtenu de l'exécution de "Visited = Planified" ou de l'heuristique basée sur le CSP.

Paramétrage

Les paramètres choisis sont :

- **Température initiale** $\theta = 100$
- **Coefficient de décroissance de la température** $\alpha = 0.95$
- **Température finale** $\theta_f = 1$
- **Nombre d'itérations pour un palier** $\imath_r = 3 \times N$ où N est le nombre de sommets.
- **Critère d'arrêt** $\theta \leqslant \theta_f$

4.4.2 Recherche à Voisinages Variables (RVV)

Toute cette méthode se base sur la construction de $kmax$ voisinages. Pour notre problème, nous avons emprunté trois ($kmax = 3$) des voisinages déjà définis :

Voisinages

- $\mathcal{N}_1$: Tous les nœuds qui sont des feuilles de l'arbre d'agrégation et qui ont le *sink* comme voisin, prendrons ce dernier comme parent en évitant de créer des cycles.
- $\mathcal{N}_2$: Soit un nœud ayant le plus de fils, ce voisinage consiste à changer, lorsque cela est possible, de parent pour ces nœuds fils.
- $\mathcal{N}_3$: Pour ce voisinage, nous utiliserons l'algorithme SDA modifié.

Recherche locale

Pour la recherche locale, nous avons choisi le recuit simulé tel qu'il a été défini dans la section précédente.

Critère d'arrêt

Après un nombre d'itérations fixé à 50, l'algorithme arrête son exécution.

5 Expérimentation numérique

5.1 Implémentation

5.1.1 Ubuntu

Ubuntu est un système d'exploitation libre, conçu initialement pour les ordinateurs de bureau et portables. Son nom provient d'un ancien mot, bantou, signifiant littéralement "Je suis ce que je suis grâce à ce que nous sommes tous". Cette distribution Linux est basée sur Debian GNU/Linux et vise à proposer un système d'exploitation pourvu d'une interface graphique, conviviale et aussi stable que Debian, mais sans l'aspect technique qui met à l'écart les nouveaux venus dans le monde Linux. Ce projet a débuté en 2004 et l'initiateur du projet n'est autre que le milliardaire sud-africain Mark Shuttleworth. Ce dernier a spécialement fondé la société Canonical pour le développement d'Ubuntu. [1]

5.1.2 Langage C

Le langage C a été mis au point par D.Ritchie et B.W.Kernighan au début des années 70. Leur but était de développer un langage qui permettrait d'obtenir un système d'exploitation de type UNIX portable. D.Ritchie et B.W.Kernighan se sont inspirés des langages B et BCPL pour créer un nouveau langage : le langage C. La première définition de ce langage a été donnée dans leur livre commun « The C programming language ». Toutefois, suite à l'apparition de nombreux compilateurs C, l'ANSI (abréviation de American National Standards Institute) a décidé de normaliser ce langage pour donner ce que l'on appelle le C-ANSI. Suite à cette norme, Ritchie et Kernighan ont sorti une deuxième édition du livre en intégrant les modifications apportées par l'ANSI. Le langage C reste un des langages les plus utilisés actuellement. Cela est dû au fait qu'il soit un langage qui comporte des instructions et des structures de haut niveau, tout en générant un code très rapide grâce à un compilateur très performant. Un programme écrit en C en respectant la norme ANSI est portable sans modifications sur n'importe quel système d'exploitation disposant d'un compilateur C : Windows, UNIX, VMS (système des VAX) ou encore OS/390 ou z/Os (l'OS des mainframes

1. Source : https ://www.generation-nt.com/test-presentation-ubuntu-linux-kubuntu-gnome-kde-jaunty-jackalope-xfce-edubuntu-usbuntu-mythbuntu-article-854681-1.html

IBM). La rapidité des programmes écrits en C est en grande partie due au fait que le compilateur présuppose que le programmeur sait ce qu'il fait : il génère un code ne contenant pas de vérifications sur la validité des pointeurs, l'espace d'adressage, etc. Ainsi, les programmes en C sont très compacts. [2]

5.1.3 Sublime Text

Sublime Text est un éditeur de texte vraiment puissant. Disponible à la fois sur Windows, sur Mac et sur Linux, il est conçu pour prendre en charge plusieurs langages de programmation variés allant du langage de programmation C à l'Action Script en passant par les langages PHP, Objective-C ou encore OCaml, voire même du Scripting comme le Shell Scripting ou encore le SQL. Ce qui fait réellement sa force est sa capacité à prendre en charge de nombreux langages mais aussi d'apporter de nombreuses fonctionnalités pratiques qui faciliteront la création de code pour les développeurs. [3]

Pour pouvoir tester les méthodes proposées, nous avons :

- Généré, en premier lieu, une permutation aléatoire ;
- Utilisé cette permutation pour créer un arbre couvrant aléatoire ;
- Ajouté un nombre aléatoire d'arêtes entre des sommets pris aléatoirement deux à deux, dans le cas où l'arête n'existe pas déjà dans l'arbre couvrant ;
- Choisi un *sink* aléatoirement, puis effectué tous les changements nécessaires pour que ce *sink* soit représenté par l'indice 0.

Toutes ces étapes nous ont permis de générer aléatoirement un réseau de capteurs sans fil.

Nous avons par la suite implémenté un algorithme permettant de donner un arbre des plus courts chemins enraciné au *sink*. Cet algorithme fonctionne comme suit :

En commence d'abord par relier le *sink* à tous ses voisins. Pour chacun des sommets restants, on cherche si l'un de ses voisins est un voisin du *sink*... Ainsi de suite, jusqu'à construire un arbre des plus courts chemins enraciné au *sink*.

L'arbre est utilisé comme entrée dans l'exécution de l'algorithme SDA, nous obtenons alors dans un premier temps, la latence résultante de ce protocole. En ne modifiant que quelques lignes de code du protocole SDA, nous obtenons facilement le protocole SDA modifié.

Nous avons également programmé les deux heuristiques proposées, **"Visited = Planified"** et **celle basée sur le CSP**.

5.2 Présentation des résultats et commentaires

Dans la présentation des résultats de l'exécution des algorithmes implémentés, nous allons utiliser les abréviations définies dans la table 5.1.

2. Source : https ://bit.ly/2lm72QM
3. Source : https ://www.supinfo.com/articles/single/5302-sublime-text

Notation	Définitions
SDA	Protocole SDA.
SDAI	Protocole SDA modifié.
VisPlan	Heuristique "Visited = Planified".
CSPA	Heuristique basée sur le CSP.

TABLE 5.1 – Abréviations utilisées

$N = 10$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I8**	**I10**	**Latence Moyenne**
SDA	8	8	9	6	5	5	7	4	7	7	6.6
VisPlan	8	8	9	6	7	6	8	6	9	8	7.5
VisPlan et SDA	8	8	9	6	5	5	7	4	7	7	6.6
VisPlan et SDAI	7	5	6	6	5	5	5	5	8	6	5.8
CSPA	8	8	9	6	7	6	8	6	9	7	7.4
CSPA et SDAI	7	5	6	6	6	5	5	5	8	5	5.8

TABLE 5.2 – Latences obtenues pour 10 instances de 10 sommets

$N = 20$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I8**	**I10**	**Latence Moyenne**
SDA	18	6	7	7	10	14	11	12	15	9	10.9
VisPlan	18	6	8	8	12	14	12	15	17	9	11.9
VisPlan et SDA	18	6	7	7	10	14	11	12	15	9	10.9
VisPlan et SDAI	14	7	7	8	8	9	8	10	8	7	8.6
CSPA	18	6	7	8	12	15	13	14	17	9	11.9
CSPA et SDAI	14	7	7	8	7	11	9	11	10	7	9.1

TABLE 5.3 – Latences obtenues pour 10 instances de 20 sommets

$N = 30$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I8**	**I10**	**Latence Moyenne**
SDA	12	17	8	10	24	8	9	13	7	9	11.7
VisPlan	15	20	10	12	26	13	11	20	8	11	14.6
VisPlan et SDA	12	17	8	10	24	8	8	14	7	9	11.7
VisPlan et SDAI	9	12	8	9	15	9	9	11	7	9	9.8
CSPA	15	19	10	11	27	14	12	22	8	11	14.9
CSPA et SDAI	8	11	9	8	15	9	9	10	7	9	9.5

TABLE 5.4 – Latences obtenues pour 10 instances de 30 sommets

Nous avons exécuté notre programme sur 10 instances de 10, 20, 30, 40 et 50 nœuds.

Les résultats respectifs de l'exécution sont résumés dans les tables 5.2, 5.3, 5.4, 5.5 et 5.6.

Lors de l'exécution du programme, nous avons remarqué que l'hybridation entre CSPA et SDAI donne de meilleurs résultats que les autres méthodes implémentés dans le cas où le réseau est dense, c'est-à-dire qu'il comporte un grand nombre d'arêtes. Et

$N = 40$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I8**	**I10**	**Latence Moyenne**
SDA	15	33	15	20	35	14	10	38	16	11	20.7
VisPlan	22	36	23	27	36	21	13	38	20	16	25.2
VisPlan et SDA	14	34	18	21	35	15	10	38	15	12	21.2
VisPlan et SDAI	11	22	12	12	21	11	10	27	12	10	14.8
CSPA	22	35	23	26	36	22	12	38	22	16	25.2
CSPA et SDAI	11	21	12	13	21	10	10	27	11	10	14.6

TABLE 5.5 – Latences obtenues pour 10 instances de 40 sommets

$N = 50$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I8**	**I10**	**Latence Moyenne**
SDA	16	44	9	8	19	9	27	42	29	9	21.2
VisPlan	22	45	10	10	31	11	37	43	38	12	25.2
VisPlan et SDA	17	44	9	9	20	9	29	42	28	9	21.6
VisPlan et SDAI	11	29	9	9	15	9	19	26	15	10	15.2
CSPA	22	45	10	11	32	10	36	43	36	12	25.7
CSPA et SDAI	13	29	9	9	15	9	20	37	15	11	16.7

TABLE 5.6 – Latences obtenues pour 10 instances de 50 sommets

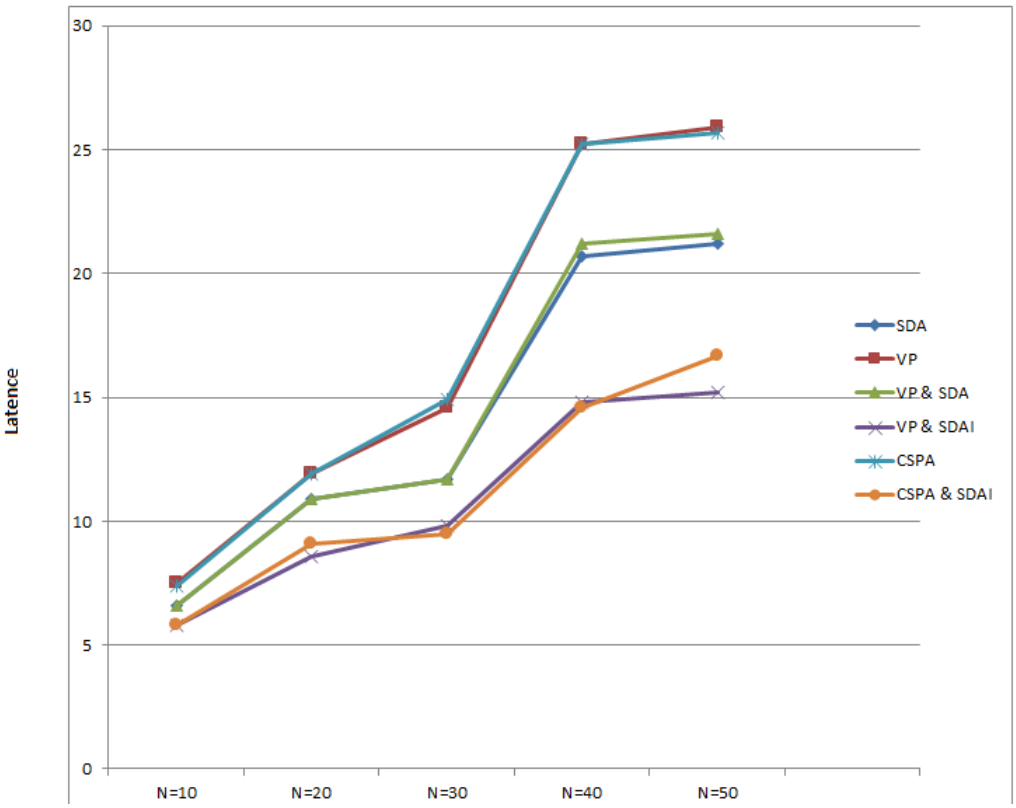


FIGURE 5.1 – Latence en fonction du nombre de nœuds

ce, malgré le fait que, généralement, la latence obtenue par CSPA ne soit pas bonne.

Ceci s'explique comme suit :

Dans l'arbre d'agrégation construit par CSPA, un même nœud a beaucoup de fils. Et puisque le réseau est dense (en terme de nombre d'arêtes), l'application de SDAI permet de planifier plusieurs feuilles d'un même niveau au même *slot*, et donc de réduire considérablement la latence.

La figure 5.1 montre que le protocole SDA et l'hybridation entre VisPlan et SDA donnent presque la même latence.

La latence résultante de l'exécution des deux protocoles CSPA et VisPlan est presque égale, comme le montrent leurs courbes respectives.

Nous rajouterons à ces comparaisons, les résultats de l'exécution de l'hybridation entre RVV et RS. Ces deux méthodes sont en cours d'implémentation, et n'ont pas été achevées par faute de temps.

Conclusion

Arrivé au terme de ce travail, il y a lieu de rappeler l'essentiel de ce qui a été réalisé, les points saillants ainsi que les principaux résultats.

Il s'agissait dans ce projet d'étudier l'agrégation des données dans les réseaux de capteurs sans fil, en vue d'adapter des schémas d'optimisation approchée (heuristiques et métaheuristiques) pour minimiser la latence.

Après un état de l'art sur les réseaux de capteurs sans fil, notamment en matière de protocoles et autres approches existants sur l'agrégation des données dans les réseaux de capteurs sans fil. Nous avons, dans un premier temps, suivi le cheminement naturel d'une étude de Recherche Opérationnelle :

- Modélisation du problème d'agrégation des données dans les réseaux de capteurs sans fil.
- Étude et analyse des différents protocoles existants

Ces deux points principaux, nous ont permis de proposer une modélisation utilisant le concept efficace de CSP et de concevoir un algorithme "Visited = Planified" et un algorithme basé sur le CSP.

Nos algorithmes choisis et proposés ont été implémentés et testés sur une batterie d'instances.

Les résultats obtenus améliorent substantiellement la qualité des solutions.

Ces performances sont mesurés par les indicateurs suivants :

- Amélioration de la latence.
- Mise à disposition de solutions diversifiées et leur utilisation dans l'adaptation de méthodes sophistiquées telles que les métaheuristiques.

En continuation avec notre projet, les perspectives qui s'annoncent seraient d'utiliser des solvers (tels que AMPL) permettant de résoudre le modèle CSP dédié à l'agrégation des données dans les réseaux de capteurs sans fil, ou encore faire appel aux métaheuristiques dans la résolution de ce même modèle.

Il serait intéressant de passer aux implémentations parallèles et distribuées pour le traitement en blocs des réseaux de capteurs sans fil exploitant des milliers de capteurs.

Bibliographie

[Akyildiz et al., 2002] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks : a survey. *Computer networks*, 38(4) :393–422.

[Apt, 2003] Apt, K. (2003). *Principles of constraint programming*. Cambridge university press.

[Ba et al., 2015] Ba, M., Flauzac, O., Niang, I., and Nolot, F. (2015). Routage et agrégation de données dans les réseaux de capteurs sans fil structurés en clusters auto-stabilisants. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, 21 :85–107.

[Bagaa et al., 2014] Bagaa, M., Challal, Y., Ksentini, A., Derhab, A., and Badache, N. (2014). Data aggregation scheduling algorithms in wireless sensor networks : Solutions and challenges. *IEEE Communications Surveys & Tutorials*, 16(3) :1339–1368.

[Bagaa et al., 2012] Bagaa, M., Derhab, A., Lasla, N., Ouadjaout, A., and Badache, N. (2012). Semi-structured and unstructured data aggregation scheduling in wireless sensor networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 2671–2675. IEEE.

[Berge and Minieka, 1973] Berge, C. and Minieka, E. (1973). Graphs and hypergraphs.

[Bondy et al., 1976] Bondy, J. A., Murty, U. S. R., et al. (1976). *Graph theory with applications*, volume 290. Citeseer.

[Bories-Longuet and Alfonsin, 2015] Bories-Longuet, F. and Alfonsin, J. R. (2015). *Graphes et Combinatoire*. Ellipses.

[Chen et al., 2005] Chen, X., Hu, X., and Zhu, J. (2005). Minimum data aggregation time problem in wireless sensor networks. In *International conference on mobile ad-hoc and sensor networks*, pages 133–142. Springer.

[Cui, 2016] Cui, J. (2016). *Data Aggregation in Wireless Sensor Networks*. PhD thesis, INSA Lyon.

[Huang et al., 2007] Huang, S.-H., Wan, P.-J., Vu, C. T., Li, Y., and Yao, F. (2007). Nearly constant approximation for data aggregation scheduling in wireless sensor networks. In *INFOCOM 2007. 26th IEEE international conference on computer communications. IEEE*, pages 366–372. IEEE.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598) :671–680.

[Malhotra et al., 2011] Malhotra, B., Nikolaidis, I., and Nascimento, M. A. (2011). Aggregation convergecast scheduling in wireless sensor networks. *Wireless Networks*, 17(2) :319–335.

[Maraiya et al., 2011] Maraiya, K., Kant, K., and Gupta, N. (2011). Wireless sensor network : a review on data aggregation.

[Mladenović and Hansen, 1997] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11) :1097–1100.

[Sakarovitch, 1984] Sakarovitch, M. (1984). *Optimisation combinatoire : méthodes mathématiques et algorithmiques.[1], Graphes et programmation linéaire*. Hermann.

[Tian et al., 2011] Tian, C., Jiang, H., Wang, C., Wu, Z., Chen, J., and Liu, W. (2011). Neither shortest path nor dominating set : aggregation scheduling by greedy growing tree in multihop wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 60(7) :3462–3472.

[Yagouni et al., 2015] Yagouni, M., Mobasti, Z., Bagaa, M., and Djaoui, H. (2015). Contribution to the optimization of data aggregation scheduling in wireless sensor networks. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 235–245. Springer.

[Yick et al., 2008] Yick, J., Mukherjee, B., and Ghosal, D. (2008). Wireless sensor network survey. *Computer networks*, 52(12) :2292–2330.

[Yu et al., 2009] Yu, B., Li, J., and Li, Y. (2009). Distributed data aggregation scheduling in wireless sensor networks. In *INFOCOM 2009, IEEE*, pages 2159–2167. IEEE.