

ACADÉMIE DE MONTPELLIER
UNIVERSITÉ MONTPELLIER II
— SCIENCES ET TECHNIQUES DU LANGUEDOC —

MÉMOIRE DE STAGE DE MASTER

SPÉCIALITÉ : **Recherche en Informatique**
Mention : **Informatique, Mathématiques, Statistiques**

effectué au LIRMM/INFO

—
sous la direction de RODOLPHE GIROUDEAU ET JÉRÔME PALAYSI

Routage dans les réseaux radio

par

Florent Charre

Soutenu le 19 juin 2007

Remerciements

Je tiens à remercier chaleureusement Rodolphe Giroudeau et Jérôme Palaysi pour leur encadrement encadrement de qualité, leur aide et leurs conseils qu'ils m'ont procurés tout le long de mon stage. Je remercie également Benoît Darties pour ses précieux conseils et encouragements durant ce stage et particulièrement lors de la rédaction de ce mémoire ainsi que l'ensemble de l'équipe APR du LIRMM pour leur accueil. Mes remerciements vont également à mes amis stagiaires de l'école doctorale pour m'avoir aidé et supporté pendant ces 6 mois de stage. Enfin je remercie ma famille et mes amis pour m'avoir soutenu depuis le début de mes études.

Résumé

Dans ce document nous abordons un problème induit par le modèle utilisé par les réseaux radio. Notre objectif consiste à satisfaire une collection de requêtes de communication dans un environnement où les nœuds du réseau ne sont pas forcément à portée directe de communication, et dans lequel les émissions simultanées de nœuds trop proches induisent des zones de brouillage. Dans un premier temps nous formalisons le problème étudié, et annonçons sa difficulté selon la topologie du réseau. Nous ciblerons ensuite notre étude sur la chaîne, topologie sur laquelle le problème reste encore ouvert. Nous donnerons à cet effet deux algorithmes d'approximation polynomiaux, ainsi qu'une stratégie (exponentielle en temps) permettant de calculer de manière exacte une solution optimale. Nous analyserons alors les performances de nos heuristiques par rapport à la solution optimale dans la dernière partie.

Table des matières

1	Introduction	3
1.1	Présentation du problème DAWN, modèle utilisé	4
1.2	Définitions, formalisme	5
1.2.1	Les deux problèmes DAWN	6
1.2.2	Notation des problèmes	9
1.3	État de l'art	9
2	DAWN dans les arbres	11
2.1	Analyse du problème $DP arbres, 1 D_{min}$	11
2.2	Construction du graphe des conflits	12
2.3	Algorithme sur les arbres de charge 1	13
3	DAWN sur les chaînes	15
3.1	Propriétés spécifiques aux chaînes	15
3.2	Le problème $DP Chaîne, uni, 1 D_{min}$	16
3.3	Analyse du problème $DP Chaîne, uni D_{min}$	17
3.4	Heuristiques sur la chaîne	17
3.4.1	Algorithme de balayage GD	17
3.4.2	Algorithme de balayage DG	23
3.5	Lemmes - Propriétés	24
4	Performances	26
4.1	Programmation par contraintes	26
4.1.1	Modèle	26
4.1.2	Analyse de l'algorithme	28
4.2	Performances des algorithmes	28
5	Conclusion et perspectives	30
A	Annexes	32
A.1	Efficacité des algorithmes	32
A.2	Encore un algorithme sur la chaîne	34
A.2.1	Algorithme	34

Table des figures

1.1	Δ -port-émission	4
1.2	1-port-réception	4
1.3	Exemple de modélisation d'une instance DAWN	6
1.4	DAWN-paths / DAWN-requests	7
1.5	DAWN-paths / DAWN-requests	8
2.1	Instance DAWN dans un arbre	11
2.2	Graphe des conflits	12
2.3	Assignation de dates	14
3.1	Contre-exemple monotonie	15
3.2	Preuve chaînes - 1	17
3.3	Preuve chaînes - 2	17
3.4	Algorithme GD - 1	17
3.5	Preuve extrémités - 1	24
3.6	Preuve extrémités - 2	25
3.7	Preuve extrémités - 3	25
A.1	Application de l'algorithme GD	32
A.2	Application de l'algorithme DG	32
A.3	Application de l'algorithme GD	33
A.4	Application de l'algorithme DG	33

Chapitre 1

Introduction

Le développement important des réseaux sans-fil pose de nouveaux problèmes algorithmiques et combinatoires, notamment liés aux spécificités du médium radio. Ces problèmes cherchent à satisfaire des requêtes de communication¹ dans des réseaux où les nœuds peuvent communiquer sans nécessairement être à portée directe de communication (on parle de réseaux multi-sauts) et où les émissions simultanées de nœuds trop proches génèrent des zones de brouillage dans lesquelles aucune réception n'est possible. Ces réseaux peuvent être supervisés, auquel cas et pour satisfaire une requête, le système assigne une route qui correspond aux différents nœuds par lesquels le message doit transiter pour arriver à destination. Si l'on suppose également que les émissions sont synchrones, il convient de définir pour chacun des nœuds d'une route une date à laquelle émettre le message. On peut choisir de temporiser l'émission sur certains de ces nœuds afin d'éviter des brouillages avec d'autres requêtes. Étant donnée une collection de requêtes, l'objectif est de minimiser le temps nécessaire pour satisfaire toutes les requêtes. Il existe deux familles principales de problèmes, l'une dans laquelle les routes sont données (DAWN-paths) et l'autre dans laquelle elles sont à trouver (DAWN-requests).

L'étude de ce modèle idéal est motivée par la possibilité qu'il donne de mesurer les performances de protocoles réels (existants ou futurs) de communication point à points dans les réseaux sans-fil (par exemple les réseaux utilisant la norme HIPERLAN/2).

Dans ce rapport nous évaluons la complexité du problème selon la topologie du réseau (arbres, chaînes...) ainsi que d'autres paramètres de la donnée du problème dans les chapitres 2 et 3. Puis nous proposerons des algorithmes polynomiaux d'approximation pour le problème dans les chaînes dans la deuxième partie du chapitre 3. Une étude théorique nous permettra d'établir une borne supérieure dans le pire des scénarios.

Dans le chapitre 4 nous verrons au travers d'expérimentations que le coût d'une solution calculée à partir des heuristiques d'approximations s'avère généralement bien meilleure que la borne calculée théoriquement. Nous montrerons également que ces résultats expérimentaux sont très proches de la solution optimale, calculée à partir d'une méthode (exponentielle en temps) que nous aurons proposés. Ces résultats encourageants laissent la porte ouverte à d'éventuels al-

¹On parle de requête de communication lorsqu'un nœud veut envoyer un message à un autre nœud du réseau.

algorithmes de résolution exacte sur les réseaux dont la topologie est une chaîne, bien que l'existence de ces derniers reste encore à établir.

1.1 Présentation du problème DAWN, modèle utilisé

Nous nous intéressons à un problème algorithmique inspiré du fonctionnement des réseaux sans-fil dans le contexte suivant :

- Un nœud du réseau ne peut pas à la fois recevoir et émettre un message, mais il peut toutefois ignorer une réception pour procéder à une émission (half-duplex).
- Les émissions sont omnidirectionnelles, un message émis par un nœud atteint tous ses nœuds voisins (Δ -port-émission, Fig 1.1).
- Un nœud ne peut recevoir qu'un seul message à la fois, si plusieurs de ses voisins émettent un message en même temps, il n'en recevra aucun (brouillage) (1-port-réception, Fig 1.2).
- Le temps est divisé en étapes de longueur égales (slots), chaque émission/réception se fait dans un seul slot (réseau synchrone).
- La topologie est fixée et connue par un superviseur (routage centralisé).

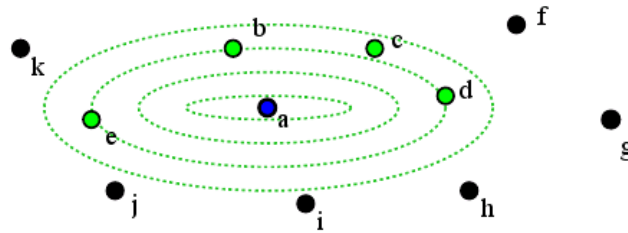


FIG. 1.1 – Lorsque le nœud a émet un message tous les sommets à portée de communication (b, c, d, e) le reçoivent.

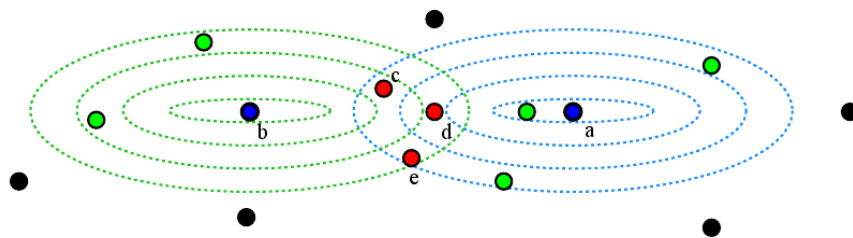


FIG. 1.2 – Si les nœuds a et b émettent un message au même instant, les sommets c, d et e ne peuvent en recevoir aucun car ils se trouvent dans la zone d'émission des deux sommets émetteurs. Les messages sont donc brouillés.

Le réseau est modélisé par un graphe orienté $G = (V, E)$, où V représente l'ensemble des nœuds du réseau et E l'ensemble des liens de communication : il existe un arc (x, y) si le nœud x peut communiquer avec le nœud y . Par la suite on considère que si un nœud peut communiquer avec un autre, alors la communication dans l'autre sens est également possible, nous représenterons donc le réseau comme un graphe orienté symétrique. Ce modèle, détaillé dans [1] est utilisé pour le problème DAWN dans [2] et pour des problèmes de diffusion (*broadcasting* en anglais) [3] ou de rassemblement (*gathering* en anglais) [4] [5]. Le problème DAWN consiste à satisfaire le plus rapidement possible une collection de requêtes de communication (une requête r est une paire de sommets (s, t) , signifiant que le nœud s veut envoyer un message au nœud t). Pour cela, on affecte à chaque nœud des dates qui correspondent aux étapes auxquelles il doit émettre les messages de chaque requête qui passe par ce nœud.

1.2 Définitions, formalisme

Définition 1.1 (Parcours) *Un parcours dans un graphe G est une liste ordonnée de sommets du graphe, telle que deux sommets consécutifs dans la liste sont adjacents dans G (terminologie utilisée dans [6]).*

Définition 1.2 (Fonction de routage) *Un graphe $G = (V, E)$ et une collection de requêtes R étant donnés, une fonction de routage est une fonction qui associe à chaque requête $r = (s, t)$ de R un parcours P_r reliant les sommets s à t .*

Définition 1.3 (Assignation de dates) *Soit (r, x) le couple associant la requête r au sommet x . Une assignation de dates est une fonction $d : (x, r) \rightarrow \mathbb{N}$ qui associe un entier positif à chaque couple sommet/requête. Ce nombre correspond à la date à laquelle le sommet x doit relayer le message de la requête r .*

Définition 1.4 (Charge) *La charge d'un sommet d'un sommet est le nombre de routes de communications contenant ce sommet. La charge (sous-entendu "d'un réseau") est le maximum des charges des sommets.*

Exemple Une charge maximale de 1 signifie qu'un sommet quelconque du graphe se situe sur le parcours d'une requête au plus.

Soient une collection de requêtes R et une fonction de routage P :

- une requête r étant donnée, $l(r)$ est le nombre de sommets de P_r moins 1.
- $s(r)$ est le sommet source de r .
- $t(r)$ est le sommet destination de r .
- $d(r, x)$ est la date à laquelle le sommet x , appartenant à P_r , transfère le message à son successeur dans P_r .
- L_{max} est la taille de la plus grande requête de la collection R .

Une assignation de date d est **correcte** si et seulement si pour toute requête r avec $P_r = (x_0, x_1, \dots, x_k)$ on a $d(r, x_0) < d(r, x_1) < \dots < d(r, x_k)$. Elle est **sans conflit** si et seulement si, lorsque $d(r, x_i) = d(r', y_j)$ les trois propriétés ci-dessous sont respectées :

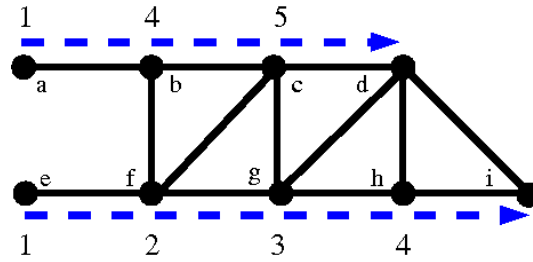


FIG. 1.3 – Cette figure représente un réseau et deux requêtes de communication. Le premier message va de a vers d en passant par les sommets b et c , le deuxième message va de e vers i en passant par les sommets f , g et h . Les dates d'émission de chaque nœud sont représentées le long des routes.

1. $x_i \neq y_j$
2. $x_{i+1} \neq y_j$ et $y_{j+1} \neq x_i$
3. (x_i, y_{j+1}) et $(y_j, x_{i+1}) \notin E(G)$

Les trois points ci-dessus reprennent le modèle présenté dans la section 1.2 :

- La première condition permet d'éviter le multiplexage : un nœud ne peut pas émettre deux messages à la même étape.
- La deuxième condition signifie qu'un nœud ne peut pas émettre et recevoir un message en même temps. On peut toutefois noter qu'un nœud peut ignorer la réception d'un message pour procéder à une émission.
- Enfin, la dernière condition modélise la Δ -port émission : un nœud peut recevoir un message d'un de ses voisins seulement si aucun autre de ses voisins n'émet au même moment.

Le problème DAWN consiste à trouver une assignation de dates correcte et sans conflit pour les différentes requêtes dans un réseau en minimisant la plus grande date affectée. Une instance du problème est illustrée sur la figure 1.3. On distingue deux types de problèmes : DAWN-paths, si les routes de communication sont imposées par une politique de routage, et DAWN-requests si elles ne le sont pas. Nous vous présentons formellement ces deux problèmes algorithmiques dans la section suivante.

1.2.1 Les deux problèmes DAWN

Le problème DAWN-paths

Une collection de requêtes de communication à satisfaire étant donnée, ce problème se propose de trouver une assignation de dates correcte et sans conflit le long des routes de communications.

Données : Un graphe G , une collection de requêtes r_i , un entier naturel D , une fonction de routage P .
Question : Peut-on satisfaire l'ensemble de ces requêtes en un nombre d'étapes inférieur ou égal à D ?

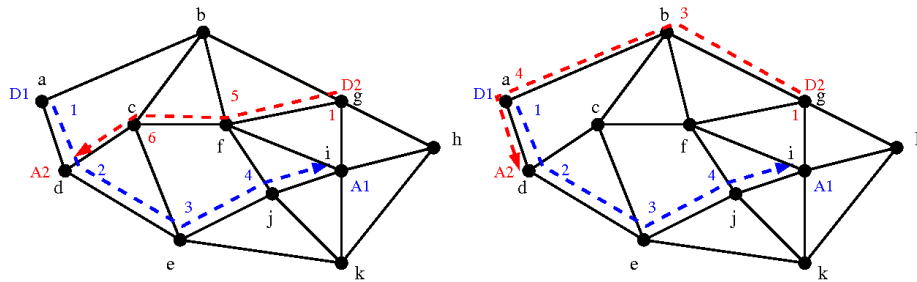


FIG. 1.4 – Considérons le réseau G ci-dessus, une collection de requêtes R contenant les requêtes $r_1 = (a, i)$ et $r_2 = (g, d)$ et la fonction de routage P telle que $P_{r_1} = (a, d, e, j, i)$ et $P_{r_2} = (g, f, c, d)$. Le figure de gauche donne une solution optimale à l’instance (G, R, P) de DAWN-paths, et le figure de droite donne une solution optimale à l’instance (G, R) de DAWN-requests. L’une en 6 étapes, et l’autre en 4.

Nous exprimons différentes déclinaisons du problème DAWN-paths :

- min-DAWN-paths, version optimisation consistant à trouver une assignation de dates qui minimise la date d’émission maximale.
- min-DAWN-K-paths, version de min-DAWN-paths dans lequel le nombre de requêtes est borné par la constante K .
- D-DAWN-paths, problème de décision dans lequel la date maximale admissible D est extraite de la donnée.

Le problème DAWN-requests

Ce problème diffère du problème DAWN-paths par l’absence de la fonction de routage : on connaît les requêtes de communication, mais contrairement à DAWN-paths, le parcours de chacune d’entre elles n’est pas fixé.

Données : Un graphe G , une collection de requêtes r_i , un entier naturel D .
Question : Peut-on satisfaire l’ensemble de ces requêtes en un nombre d’étapes inférieur ou égal à D ?

Comme pour DAWN-paths nous exprimons différentes déclinaisons du problème DAWN-requests :

- min-DAWN-requests, version optimisation consistant à trouver une assignation de dates qui minimise la date d’émission maximale.
- min-DAWN-K-requests, version de min-DAWN-requests dans lequel le nombre de requêtes est borné par la constante K .
- D-DAWN-paths, problème de décision dans lequel la date maximale admissible D est extraite de la donnée.

Ces deux types de problèmes sont illustrés sur la figure 1.4.

Remarque Il n’existe pas de routage fixe optimal pour le problème DAWN-paths. Autrement dit, on ne peut pas fixer à l’avance un parcours pour une requête et garantir que l’utilisation de ce parcours donnera une solution optimale du problème. Il faudra recalculer les routes en fonction des instances.

Preuve La remarque précédente est prouvée grâce à l'exemple suivant (figure 1.5) :

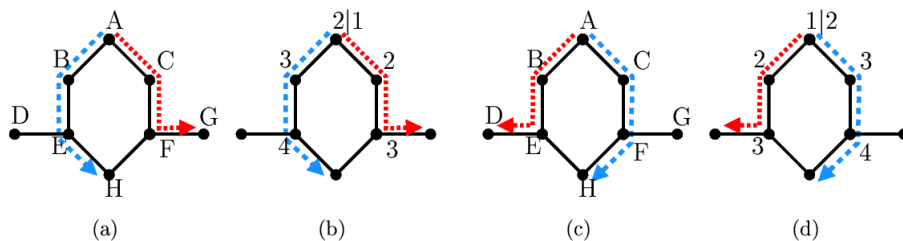


FIG. 1.5 – Sur la figure (a), nous avons une collection de requête R_a à satisfaire telle que $R_a = (r_1, r_2)$ avec $r_1 = (A, H)$ et $r_2 = (A, G)$. La figure (b) présente une assignation de date optimale pour cette collection de requêtes. Comme l'on peut le constater, $P_{r_1} = (A, B, E, H)$, tandis que $P_{r_2} = (A, C, F, G)$. La figure (c) représente une nouvelle collection de requête R_b telle que $R_b = (r_1, r_3)$ avec $r_1 = (A, H)$ et $r_3 = (A, D)$. On constate alors sur la figure (d) que la requête r_1 va devoir choisir un parcours différent de celui utilisé pour de la première collection de requête R_1 si l'on veut garantir l'optimalité de la solution pour cette instance.

□

1.2.2 Notation des problèmes

Pour synthétiser la formulation et faciliter la compréhension des problèmes nous avons introduit une notation, inspirée de celle des problèmes d'ordonnement proposée par Graham et al. [7] et par Blazewicz et al. [8]. Pour définir un problème *DAWN* nous utilisons trois paramètres α, β, γ .

Le paramètre α définit la nature du problème :

- $\alpha = DP$ signifie que l'on parle du problème *DAWN-paths*
- $\alpha = DR$ signifie que l'on parle du problème *DAWN-requests*.

Le paramètre β définit la topologie du réseau, le nombre de requêtes par sommets... ($\beta = \beta_1\beta_2\dots$) :

- $\beta_1 = arbre$ signifie que le graphe est un arbre...
- $\beta_1 = .$ signifie que le graphe est quelconque.
- $\beta_2 = uni$ signifie que les requêtes du problème sont toutes orientées dans le même sens (unidirectionnelles). Ce paramètre est uniquement valable lorsque la topologie du réseau permet de définir un sens à une requête, c'est à dire dans les chaînes et les arbres enracinés.
- $\beta_3 = n$ signifie que la charge maximale sur un sommet est égale à n .
- $\beta_3 = .$ signifie que la charge n'est pas donnée, ou non bornée.

Le paramètre γ définit la fonction objectif :

- $\gamma = D_{min}$ signifie que l'on veut minimiser la date maximale d'émission d'une requête (problème d'optimisation).
- $\gamma = D$ signifie que l'on veut savoir si l'on peut satisfaire l'ensemble des requêtes en D dates (problème de décision).

Quelques exemples :

- $DP|chaînes, uni, 2|D_{min}$ doit être lu : « En connaissant les parcours de requêtes, le problème est de minimiser la date d'émission maximale sur un problème dans les chaînes, avec les requêtes toutes orientées dans le même sens, et de charge maximale égale à 2 ».
- $DR|bi|5$ signifie « Peut-on satisfaire les requêtes du problème dans un graphe quelconque en 5 dates, en trouvant le meilleur parcours ? »

1.3 État de l'art

Voici plusieurs résultats qui ont été prouvés par Jérôme Palaysi, Benoît Dardies et Sylvain Durand ont prouvé dans [2].

Theoreme 1.3.1 *Le problème min-DAWN-paths est NP-Difficile et non-approximable à une constante multiplicative près, même restreint à des parcours de taille 1. Pour tout entier naturel $D \geq 3$, le problème de décision D-DAWN-paths est NP-Complet même restreint à des parcours de taille 1.*

Theoreme 1.3.2 *Pour tout entier naturel K , le problème min-DAWN-K-paths est polynomial.*

Theoreme 1.3.3 *Le problème de décision 2-DAWN-paths est polynomial.*

DAWN-Paths :		Complexité :
Min-DAWN-Paths		NP-Difficile, non approximable à une constante multiplicative
D-DAWN-Paths	$D \leq 2$	Polynomial
	$D \geq 3$	NP-Complet
DAWN-K-Paths		Polynomial : algorithme en $O(n^K)$

DAWN-Requests :		Complexité :
Min-DAWN-Requests		NP-Difficile, non approximable à une constante multiplicative
D-DAWN-Requests	$D \leq 1$	Polynomial
	$D \geq 2$	NP-Complet
DAWN-K-Requests		Polynomial : algorithme en $O(n^K)$

TAB. 1.1 – Tableaux récapitulatifs des résultats de complexité déjà connus pour le problème DAWN.

Theoreme 1.3.4 *Le problème d'optimisation min-DAWN-requests est NP-Difficile et non approximable à une constante multiplicative près. Pour tout entier naturel $D \geq 3$, le problème de décision D-DAWN-requests est NP-Complet.*

Theoreme 1.3.5 *Le problème de décision 2-DAWN-requests est NP-Complet.*

Theoreme 1.3.6 *Le problème de décision DAWN-K-requests est polynomial.*

Preuve Les preuves des Théoremes 1.4.1 à 1.4.6 se trouvent dans [2].

Le tableau 1.1 rassemble tous ces résultats.

Theoreme 1.3.7 *Le problème DAWN-paths est NP-Complet même lorsque le graphe du réseau est un arbre.*

Preuve La preuve de ce théorème est présentée dans [9].

Comme nous pouvons le constater, les études menées jusqu'à présent étaient plutôt centrées sur la nature des instances (nombre de requêtes, taille des requêtes etc...). Le théorème 1.3.7 ci-dessus se propose toutefois d'analyser la complexité du problème différemment, en se basant sur la topologie du réseau. C'est dans ce sens que mon travail s'est orienté, tout d'abord en cherchant si le problème dans les arbres devient polynomial lorsque la charge est limitée, puis en étudiant la complexité du problème sur de nouvelles topologies.

Chapitre 2

DAWN dans les arbres

2.1 Analyse du problème $DP|arbres, 1|D_{min}$

Comme nous l'avons vu précédemment, le problème $DP|arbres|D_{min}$ (minimiser le nombre d'étapes pour satisfaire une collection de requêtes dans les arbres) est NP-Difficile. Nous nous sommes donc intéressés au problème dans les arbres, lorsque la charge maximale est bornée à 1. Nous sommes capable de résoudre ce problème à une unité de la solution optimale en temps polynomial, ce qui rappelle fortement le problème de coloration des arêtes dans les graphes. Voici un exemple pour illustrer le problème.

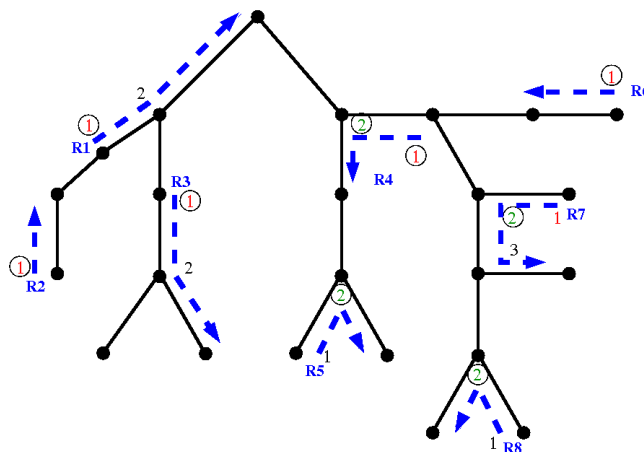


FIG. 2.1 – Une instance du problème $DP|arbres, 1|D_{min}$. Huit requêtes sont réparties sur l'arbre G , et les dates sont affectées de manière gloutonne, ce qui engendre des conflits (entourés sur l'image).

2.2 Construction du graphe des conflits

Définition 2.1 Deux requêtes r_k et r_l sont **potentiellement en conflit** si et seulement si : $\forall v_i, v_j \in V(G) (i \neq j)$ avec $v_i \in r_k$ et $v_j \in r_l$ ($r_k, r_l \in R, k \neq l$), $\exists x \in V(G)$ tel que $(v_i, x), (v_j, x) \in E(G)$ avec $x \in r_k \oplus$ ¹ $x \in r_l$. Elles sont **effectivement en conflit** si en plus des conditions précédentes, $d(r_k, v_i) = d(r_l, v_j)$.

Pour résoudre ce problème, nous construisons un graphe des conflits G_c selon le modèle suivant :

- Une requête de l'arbre G devient un sommet du graphe G_c .
- Il y a une arête entre deux sommets de G_c si deux requêtes sont potentiellement en conflit dans G .
- On étiquette les arêtes de G_c avec 2 nombres : les numéros des sommets qui sont potentiellement en conflit dans les requêtes de G . Par exemple l'arête (i, j) sera étiquetée avec $(2, 3)$ si les sommets 2 et 3 des requêtes i et j sont potentiellement en conflit dans G .

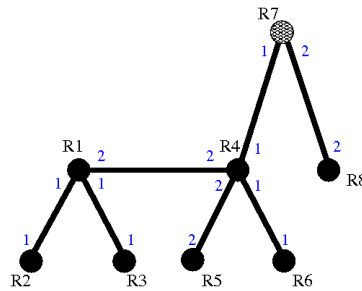


FIG. 2.2 – Graphe des conflits G_c . Le sommet correspondant à la plus grande requête est représenté par un rond hachuré. Les nombres représentent les numéros des sommets (et donc les dates) des requêtes potentiellement en conflit dans le graphe G .

Lemme 2.2.1 *Le graphe des conflits G_c d'un arbre G est également un arbre.*

Preuve Supposons qu'il existe un cycle (P_1, P_2, P_3) dans G_c . Cela signifie que les 3 requêtes correspondantes étaient en conflit dans G . Or si P_1 est en conflit avec P_2 , il y a un chemin pour aller d'un sommet de P_1 à un sommet de P_2 . C'est également le cas pour P_1 et P_3 ainsi que P_2 et P_3 . Il y a donc un chemin passant par des sommets de P_1, P_2 et P_3 qui forme un cycle, ce qui est impossible dans les arbres.

□

¹ « ou exclusif »

2.3 Algorithme sur les arbres de charge 1

Construction de G_c . On commence par assigner gloutonnement² les dates des requêtes sans se soucier des conflits potentiels. On construit ensuite le graphe des conflits (voir 2.2).

Parcours du graphe G_c . Pour enlever les conflits sur les requêtes de G , nous allons devoir vérifier les étiquettes des arêtes de G_c . Pour cela, nous faisons un parcours en profondeur du graphe des conflits. On choisit d'enraciner le graphe par un sommet correspondant à une requête de taille maximale L_{max} , puis on descend dans l'arborescence. Lorsque l'on arrive sur une arête étiquetée avec une paire de numéros identiques ((2,2) par exemple), on «incrémente» une de ces deux valeurs ce qui aura pour effet de décaler une ou plusieurs dates sur la requête correspondante du graphe G . Par défaut on choisit d'incrémenter la valeur correspondante au sommet vers lequel on se dirige selon notre parcours, et non celui d'où l'on vient.

Lemme 2.3.1 *S'il n'y a qu'une seule requête de taille L_{max} dans G , l'algorithme précédent renvoie une solution optimale du problème en $D_{max} = L_{max}$ dates.*

Preuve Si l'on a deux requêtes en conflit seulement, un seul décalage de date suffit à annuler le conflit. Étant donné que l'on commence notre parcours de G_c par le sommet représentant la requête la plus longue, on peut incrémenter la valeur représentant l'autre requête en étant certains de ne pas augmenter la date d'envoi maximale. On peut déduire par induction, et grâce au fait que l'on est dans un arbre et donc qu'il nous est impossible de repasser deux fois par le même sommet, que l'on peut régler tous les conflits en incrémentant, au maximum, une seule date dans chacune des requêtes de taille inférieure à L_{max} , et donc garantir l'optimalité.

□

Lemme 2.3.2 *L'algorithme précédent renvoie une solution égale à la date optimale plus 1 dans le cas général ou il y a plus d'une requête de taille maximale.*

Preuve Comme on l'a vu précédemment on augmente la date maximale d'émission de chaque requête de 1 date au maximum. Si une des requêtes situées sur le parcours a une taille égale à L_{max} alors la solution sera égale à $D_{max} = L_{max} + 1$.

□

²On assigne la date i au $i_{ème}$ sommet de chaque requête.

Chapitre 3

DAWN sur les chaînes

Étant donnée la difficulté du problème dans les arbres, nous nous sommes tournés vers une classe d'arbres particulière, les chaînes.

3.1 Propriétés spécifiques aux chaînes

Remarque Sur les chaînes, il n'y a qu'un seul parcours possible par requête, les problèmes DAWN-paths et DAWN-requests sont donc équivalents.

Définition 3.1 (Monotonie) Une assignation est dite *monotone sur r* si, sur tout le parcours $P_r = (x_0, x_1, \dots, x_i, \dots, x_k)$ on a :

1. $d(r, x_0) = t$
2. $d(r, x_i) = t + i, \forall i \in [0, k]$

Autrement dit, une fois partie, la progression de la requête n'est plus stoppée et se propage sur P_r jusqu'à son dernier élément. Seule la date d'émission de son sommet source est ici nécessaire puisqu'elle permet de déduire les dates d'émission des autres sommets du parcours.

Remarque Comme l'on peut le voir sur la figure 3.1, il existe des instances dans lesquelles la solution optimale ne peut pas être atteinte avec la monotonie.

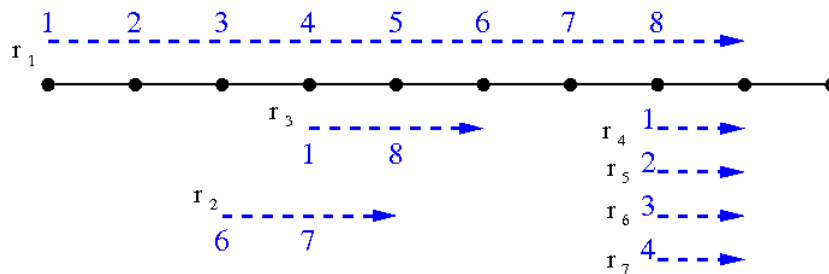


FIG. 3.1 – Dans cette instance, si l'on veut obtenir une solution optimale, la requête r_3 devra être temporisée.

3.2 Le problème $DP|Cha\hat{c}haine, uni, 1|D_{min}$

Dans ce problème, nous considérerons que tout ensemble de requêtes $R = \{r_1, r_2, \dots, r_n\}$ est ordonné tel que $t(r_i) < s(r_{i+1})$.

Définition 3.2 *Un intervalle unitaire $I = \{r_i, r_{i+1}, \dots, r_j\}$ dans un ensemble de requêtes R est un intervalle de requêtes tel que :*

- toutes les requêtes de I sont de longueur 1,
- tout sommet i tel que $i \in [s(r_i), t(r_j)]$ appartient à une requête r de I .

Un intervalle unitaire I est maximal s'il n'existe aucun autre intervalle unitaire I' de R tel que $I \subset I'$.

Theoreme 3.2.1 *$DP|Cha\hat{c}haine, 1|D_{min}$ est polynomial.*

Preuve On supposera que la plus grande requête est de longueur au moins 2 (sinon il est facile de vérifier que l'instance peut être satisfaite en une ou deux dates). Un algorithme possible consiste à satisfaire d'abord les requêtes de longueur 2 ou plus en assignant la première date possible : autrement dit, quelle que soit la requête r de longueur 2 ou plus, $d(r, s(r) + i) = i + d(r, s(r))$ pour tout $i \in [s(r), t(r)]$. À ce stade, la plus grande date assignée par d est égale à la longueur de la plus longue requête. L'algorithme assigne ensuite les dates pour les requêtes de taille 1 : considérons un intervalle unitaire maximal $I = r_i, r_{i+1}, \dots, r_j$ de R .

1. Si $t(r_{i-1}) + 1 < s(r_i)$ alors on peut satisfaire les requêtes de I avec les dates 1 et 2 au plus en affectant de manière gloutonne les dates depuis la requête r_j vers la requête r_i (i.e de droite à gauche) (Fig 3.2).
2. Si $t(r_j) < s(r_{j+1}) - 1$, nous sommes dans le cas symétrique du précédent, il faut alors affecter gloutonnement les dates de la requête r_i à la requête r_j (i.e de gauche à droite).
3. Dans les autres cas, la requête r_i est précédée d'une requête de taille supérieure ou égale à 2 et donc $d(r_{i-1}, t(r_{i-1}) - 1) \geq 2$, et r_j précède une requête de longueur supérieure ou égale à 2 et la date assignée au sommet $s(r_{j+1})$ est 1, alors :
 - Si le nombre de requêtes de I est pair alors on peut satisfaire les requêtes de I avec les dates 1 et 2.
 - Si le nombre de requêtes de I est impair, elles peuvent être satisfaites avec les dates 1, 2 et 3 et on peut vérifier que toute autre assignation de date d est telle que $\max(t) \geq 3$ (Fig 3.2).

Note : Ceci est valable dans le cas où les requêtes sont orientées de gauche à droite. Lorsque les requêtes vont de la droite vers la gauche il faut inverser les sommets s et t .

□

L'algorithme de la preuve précédente nous montre directement la propriété suivante :

Propriété 3.2.2 *Il existe une assignation de date optimale telle que :*

$$\forall r \in R, \text{ si } l(r) > 1 \text{ alors } \forall i \in [1, l(r)], d(r, s(r) + i) = i$$

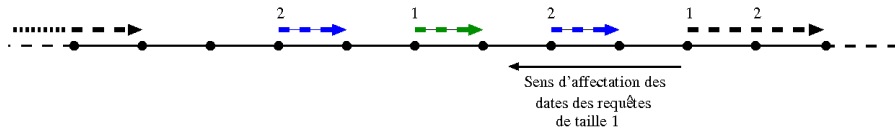


FIG. 3.2 – Pour éviter d’avoir une date maximale égale à 3 aux requêtes de taille 1, on assigne les requêtes en partant de la droite.



FIG. 3.3 – Ici nous ne pouvons pas affecter les requêtes de taille 1 avec 2 dates seulement.

3.3 Analyse du problème $DP|Chaîne, uni|D_{min}$

Ce problème s’est avéré très difficile, et malgré plusieurs tentatives de preuve de NP-complétude, basées sur des réductions SAT, des problèmes de coloration, ou de partitions, nous ne sommes pas parvenus à prouver la difficulté du problème. Nous n’avons pas trouvé non plus d’algorithme exact polynomial, je vais donc vous proposer différentes heuristiques d’approximation qui résolvent le problème sur la chaîne en temps polynomial.

3.4 Heuristiques sur la chaîne

3.4.1 Algorithme de balayage GD

Idée : Nous avons proposé un algorithme pour résoudre le problème en temps polynomial. Cet algorithme consiste à parcourir notre chaîne dans les sens des requêtes, et affecter les dates de manière gloutonne à chaque départ de requête rencontré. La première requête affectée est donc la première requête sur la chaîne. Dans le cas où plusieurs requêtes démarrent sur le même sommet, on choisit de commencer par la plus longue. Chaque requête est ainsi affectée de manière monotone, c’est à dire qu’il n’y a pas de sauts dans les dates (propriété 3.2.2).

Nous allons illustrer cela en utilisant l’instance représentée par la figure 3.4.

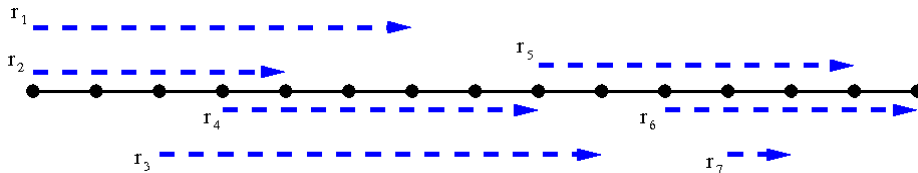


FIG. 3.4 – Sur cette chaîne à 15 sommets, nous avons une collection de requêtes $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ avec $r_1 = (1, 7)$, $r_2 = (1, 5)$, $r_3 = (3, 10)$, $r_4 = (4, 9)$, $r_5 = (9, 14)$, $r_6 = (11, 15)$ et enfin $r_7 = (12, 13)$.

14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
6	-	-	-	-	-	E	-	-	-	-	-	-	-	-	
5	-	-	-	-	E	-	-	-	-	-	-	-	-	-	
4	-	-	-	E	-	-	-	-	-	-	-	-	-	-	
3	-	-	E	-	-	-	-	-	-	-	-	-	-	-	
2	-	E	-	-	-	-	-	-	-	-	-	-	-	-	
1	E	-	-	-	-	-	-	-	-	-	-	-	-	-	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TAB. 3.1 – On commence par remplir les cases correspondantes à la 1ère requête. Le premier sommet de celle-ci émettra le message à la date 1 et ainsi de suite jusqu'à la date 6.

Représentation : On peut représenter ceci sous la forme d'une matrice, de largeur égale au nombre de sommets de la chaîne, et dont la hauteur correspondant aux dates d'affectations. La matrice sera remplie avec les caractères suivants :

- E : Qui représente l'émission d'un message par un sommet,
- I : qui signifie l'interdiction d'émettre un message à cette date pour ne pas qu'il y ait de brouillage causé par la réception simultanée de deux message par un même sommet,
- B : qui permet d'éviter la possibilité d'émettre à une étape car le sommet suivant ne peut pas envoyer et recevoir un message en même temps,
- R : qui permet de "réserver" une date afin de permettre la réception d'un message préalablement envoyé.

Si un sommet émet à la date 1, alors on met un 'E' dans la case correspondante de la matrice (tableau 3.1).

Il faut ensuite représenter les interdictions pour éviter les conflits induits par ces émissions. En effet si le premier sommet émet à la date 1, alors les sommets en conflit ne pourront pas émettre à cette même date. Un sommet ne pouvant pas envoyer et recevoir un message en même temps, le sommet 2 ne pourra pas émettre, et le sommet 3 devra également attendre, sinon il produirait un conflit car le sommet 2 recevrait deux messages en même temps. De plus le sommet 1 ne peut pas envoyer plusieurs messages à la même étape, donc les autres requêtes passant par ce même sommet devront patienter (tableau 3.2).

Pour les affectations des requêtes suivantes, il faut choisir la première date disponible, c'est à dire la première case « vide » en partant du bas. Ainsi, le premier sommet de la 2ème requête ne pourra pas émettre avant la date 4, le 2ème sommet à 5 etc...(tableau 3.3)

14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
6	-	-	-	I	B	E	R	I	-	-	-	-	-	-	
5	-	-	I	B	E	R	I	-	-	-	-	-	-	-	
4	-	I	B	E	R	I	-	-	-	-	-	-	-	-	
3	I	B	E	R	I	-	-	-	-	-	-	-	-	-	
2	B	E	R	I	-	-	-	-	-	-	-	-	-	-	
1	E	R	I	-	-	-	-	-	-	-	-	-	-	-	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TAB. 3.2 – Représentation des dates d’émission interdites en fonction des émissions précédentes (voir la signification des caractères I,R et B ci-dessus).

Au final, nous obtiendrons donc la matrice 3.4.

14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	-	I	B	E	R	I	-	-	-	-	-	-	-	-	-
6	I	B	E	R	I	E	R	I	-	-	-	-	-	-	-
5	B	E	R	I	E	R	I	-	-	-	-	-	-	-	-
4	E	R	I	E	R	I	-	-	-	-	-	-	-	-	-
3	I	B	E	R	I	-	-	-	-	-	-	-	-	-	-
2	B	E	R	I	-	-	-	-	-	-	-	-	-	-	-
1	E	R	I	-	-	-	-	-	-	-	-	-	-	-	-
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TAB. 3.3 – Les dates d’émission de la nouvelle requête se placent en fonction des émissions et des interdictions des requêtes précédentes.

14	-	-	-	-	-	-	I	B	E	R	I	-	-	-	-
13	-	-	-	-	-	I	B	E	R	I	-	-	-	-	-
12	-	-	-	-	I	B	E	R	I	-	-	-	-	-	-
11	-	-	-	I	B	E	R	I	-	-	-	-	-	-	-
10	-	-	I	B	E	R	I	-	-	-	-	-	-	-	-
9	-	I	B	E	R	I	-	-	-	-	-	I	B	E	R
8	I	B	E	R	I	-	-	-	-	-	-	I	B	E	R
7	-	I	B	E	R	I	-	-	-	I	B	E	R	I	-
6	I	B	E	R	I	E	R	I	I	B	E	R	I	-	-
5	B	E	R	I	E	I	B	E	R	I	I	B	E	R	I
4	E	R	I	E	I	B	E	R	I	I	B	E	R	I	-
3	I	B	E	I	B	E	R	I	I	B	E	R	I	-	-
2	B	E	I	B	E	R	I	I	B	E	R	I	-	-	-
1	E	I	B	E	R	I	I	B	E	R	I	E	R	I	-
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TAB. 3.4 – Matrice finale. La date d’émission maximale est alors obtenue en regardant la plus « haute » case de la matrice affectée à 'E' (ici 14).

Lemme 3.4.1 *Cet algorithme garantit un rapport d'approximation de 6 par rapport à la solution optimale de l'instance.*

Preuve Soit :

- $\pi(i)$ = charge de la colonne i .
- Π = charge maximale.
- L = longueur de la plus longue requête.

On sait que $\max(\Pi_{max}, L) \leq OPT$.

La hauteur d'une colonne i dépend du nombre d'émissions dans celle-ci, et donc de la charge $\pi(i)$, ainsi que des émissions dans les colonnes $i-2$, $i-1$, $i+1$ et $i+2$. On peut donc borner la hauteur de la plus grande colonne par 5Π et donc la date d'émission maximale par $5\Pi + L$. On sait qu'une solution optimale a forcément $\max(\Pi, L)$ dates. On a donc :

$$\frac{Algo}{OPT} \leq \frac{5\Pi + L}{\max(\Pi, L)}$$

$$si\ L > \Pi, \frac{Algo}{OPT} = \frac{5\Pi + L}{L} = \frac{5\Pi}{L} + 1 \leq 5 + 1 \leq 6$$

$$si\ L < \Pi, \frac{Algo}{OPT} = \frac{5\Pi + L}{\Pi} = 5 + \frac{L}{\Pi} \leq 5 + 1 \leq 6$$

□

Note : Nous pensons qu'il est possible de prouver que cet algorithme a un rapport d'approximation de 2 (au lieu de 6) mais nous n'avons pas eu le temps de le prouver durant ce stage.

Implémentation : Nous nous sommes servis de cette approche de l'assignation de dates sous forme de matrice pour implémenter facilement cet algorithme (Algorithmes 1 et 2).

Algorithme 1 : Heuristique parcourant la chaîne de droite à gauche.

Données : Une collection de requêtes ordonnées.

Résultat : La date d'émission maximale de l'instance.

```

pour col = 0 à nbSom - 1 faire
  pour r = 0 à nbReq faire
    si sommetDepart[r] == col alors
      date_prec = -1;
      pour l = 0 à longueur[r] faire
        remplirMatrice(col,l,matrice)

```

Algorithme 2 : Fonction remplirMatrice()

Entrées : La matrice `matriceDates`, la colonne `col` et le sommet `l` actuellement traités.

Sorties : Les dates affectées à la requête courante.

`date = 0`

tant que `dateNonDisponible` ou `date ≤ date_prec` **faire**

`date ++`

`matriceDates[col+1][date]=r`

`date_prec = date`

`remplirInterdictions()`

Complexité De la façon dont il est implémenté, cet algorithme est en $O(n.r)$, avec n = nombre de sommets de la chaîne, et r = le nombre de requêtes.

Remarque L'algorithme tel qu'il est implémenté n'est pas polynomial en fonction de la taille de la donnée. En effet le codage optimal d'une instance est de l'ordre de $O(r \log n)$ où r est le nombre de requêtes et n le nombre de sommets de la chaîne. Nous avons ici une implémentation pseudo polynomiale, puisque de complexité $O(n.r)$. Il est cependant possible d'implémenter cet algorithme en $O(r^2)$. Nous n'avons pas retenu cette solution pour deux raisons :

1. par manque de temps : l'implémentation proposée est plus facile à mettre en place et plus compréhensible pour le lecteur
2. car nous comparons la solution obtenue à celle d'un algorithme exact mais exponentiel. Aussi, adopter une implémentation en $O(r^2)$ nous aurait certainement permis de résoudre des instances de tailles supérieures, mais sans pouvoir obtenir la solution optimale sur ces dernières.

Notons enfin que bien qu'en $O(n.r)$, notre implémentation reste beaucoup plus rapide que la résolution exacte en terme de temps d'exécution.

3.4.2 Algorithme de balayage DG

Idée : Il s'agit de la même démarche que pour l'heuristique précédente, excepté le fait que l'on parcourt la chaîne dans l'autre sens, c'est à dire dans les sens contraire des requêtes. Dans notre exemple précédent, la première requête affectée sera la requête allant du sommet 12 au sommet 13. On illustre une solution renvoyée par cet algorithme dans le tableau 3.5.

14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	-	I	B	7	R	I	-	-	-	-	-	-	-	-	-
9	I	B	7	I	B	6	R	I	5	R	I	-	-	-	-
8	-	-	I	B	6	R	I	5	R	I	-	-	-	-	-
7	-	I	B	6	R	I	5	R	I	-	I	B	3	R	I
6	I	B	6	R	I	5	R	I	-	I	B	3	R	I	-
5	B	7	R	I	5	I	B	4	R	I	3	R	I	2	R
4	7	R	I	5	I	B	4	R	I	3	R	I	2	R	I
3	I	B	5	R	I	4	R	I	-	I	B	2	R	I	-
2	B	6	R	I	4	R	I	-	I	B	2	R	I	-	-
1	6	R	I	4	R	I	I	B	3	R	I	1	R	I	-
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TAB. 3.5 – Matrice finale (les E ont été remplacés par les numéros des requêtes dans l'ordre d'affectation pour plus de clarté).

3.5 Lemmes - Propriétés

Aucun algorithme n'est meilleur qu'un autre au sens de l'efficacité. En effet, si le second algorithme est plus efficace en moyenne (voir chapitre suivant) il existe des instances pour lesquelles le premier algorithme retournera une meilleure solution que le second. L'inverse est également vrai. Ceci est illustré par les exemples présentés en annexe. Sur l'un, l'algorithme DG renvoie une meilleure solution (Fig A.1 et A.2). Sur l'autre, c'est l'inverse (Fig A.3 et A.4).

Lemme 3.5.1 *Si deux requêtes ont la même extrémité, on peut faire partir la plus courte sans dégrader l'optimalité de la solution.*

Preuve Il y a 3 cas possibles :

- Les 2 requêtes débutent à un sommet d'écart (1)
- Les 2 requêtes débutent à 2 sommets d'écart (2)
- Les 2 requêtes débutent à plus de 2 sommets d'écart (3)

(1) Soit $L1$ la taille de la plus courte requête (également notée requête 1 par la suite), et $L2$ celle de la plus longue (également notée requête 2 par la suite). On a donc $L2 = L1 + 1$. Si l'on fait partir la plus courte en premier, à l'étape i , alors la dernière date affecté pour la requête 1 sera $i + L1 - 1$. Si $L1 > 1$, la deuxième requête devra attendre $i + 2$, et se terminera donc à $i + 2 + L2 - 1 = i + 2 + L1$. Maintenant si l'on fait partir la plus longue requête en premier la plus grande date affectée sera $i + L2 - 1 = i + L1$ pour la requête 2 et $i + 3 + L1$ pour la requête 1 (Fig 3.5). Il est donc avantageux de faire partir la plus courte requête en premier. Si $L1 = 1$, il est facile de vérifier que la date d'émission maximale sera $i + 2$.



FIG. 3.5 - À gauche, on fait partir la plus courte requête en premier, la date d'affectation maximale est alors $i + L1 + 2$. Tandis qu'à droite, si l'on fait partir la plus longue requête d'abord, la plus grande date assignée est $i + L1 + 3$

(2) On a $L2 = L1 + 2$. Si l'on fait partir la plus courte en premier à l'étape i , alors la dernière date affecté pour la requête 1 sera $i + L1 - 1$. La deuxième requête devra attendre $i + 1$, et se terminera donc à $i + 1 + L2 - 1 = i + L1 + 2$. Maintenant si l'on fait partir la plus longue requête la plus grande date affectée sera $i + L2 - 1 = i + L1 + 1$ pour la requête 2. La requête 1 devra attendre $i + 5$ pour commencer à émettre, donc la date maximale d'émission de cette requête sera $i + L1 - 1 + 5 = i + L1 + 4$ (Fig 3.6).

(3) Dans ce cas, il n'y a pas de conflit entre les 2 requêtes, on peut donc les faire partir à la même étape, sans perdre l'optimalité (Fig 3.7).

□



FIG. 3.6 – À gauche, on fait partir la plus courte requête en premier, la date d'affectation maximale est alors $i + L1 + 2$. Tandis qu'à droite, si l'on fait partir la plus longue requête d'abord, la plus grande date assignée est $i + L1 + 4$

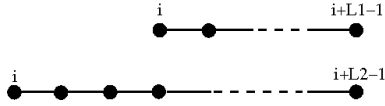


FIG. 3.7 – Les 2 requêtes peuvent partir à la même date sans conflit.

Conjecture : L'algorithme DG retourne une solution optimale du problème si toutes les requêtes de l'instance se terminent au même sommet. La preuve est basée sur une extension du lemme 3.5.1.

Chapitre 4

Performances

4.1 Programmation par contraintes

Le problème étant difficile, afin de calculer la solution optimale (minimiser la date d'envoi maximale D), nous nous sommes penchés vers la programmation par contraintes et l'utilisation du solveur CHOCO [10].

4.1.1 Modèle

Pour faciliter l'implémentation en CHOCO nous modélisons une instance du problème $DP|Chaîne, uni|D_{min}$ sous la forme d'une matrice $matriceReq[N][M]$: le nombre de ligne (N) correspond au nombre de requêtes tandis que le nombre de colonnes dans la matrice (M) correspond au nombre de sommet de la chaîne. Cette matrice est dans premier temps initialisée à 0. Puis nous affectons certains éléments de la matrice de la façon suivante :

$$matriceReq[i][j] = \begin{cases} 1 & \text{si le sommet } j \text{ est sur parcours de la requête } i \\ 0 & \text{sinon} \end{cases}$$

Par exemple, l'instance suivante, sur une chaîne de 15 sommets, composée des 10 requêtes suivantes : $r_1 = (2, 9)$, $r_2 = (3, 14)$, $r_3 = (4, 14)$, $r_4 = (8, 9)$, $r_5 = (9, 14)$, $r_6 = (9, 12)$, $r_7 = (12, 13)$, $r_8 = (13, 14)$, $r_9 = (14, 15)$ et enfin $r_{10} = (14, 15)$ sera représentée par la matrice 4.1.

Nous nous servons de cette matrice pour en créer une nouvelle, $matriceDates$, qui contiendra à la place des 1, les dates auxquelles les sommets devront effectuer l'émission. Nous déduisons de cela le modèle suivant :

Variables : $X = \{matriceDates[i][j]\}$, $\forall i \in [0, nombre_sommets]$,
 $\forall j \in [0, nombre_requêtes]$

Domaine : $\forall i \in X$, $D(i) = \{0, 1, \dots, dom\}$

1	-	1	1	1	1	1	1	1	-	-	-	-	-	-	
2	-	-	1	1	1	1	1	1	1	1	1	1	1	-	
3	-	-	-	-	1	1	1	1	1	1	1	1	1	-	
4	-	-	-	-	-	-	-	1	-	-	-	-	-	-	
5	-	-	-	-	-	-	-	-	1	1	1	1	1	-	
6	-	-	-	-	-	-	-	-	1	1	1	-	-	-	
7	-	-	-	-	-	-	-	-	-	-	-	1	-	-	
8	-	-	-	-	-	-	-	-	-	-	-	-	1	-	
9	-	-	-	-	-	-	-	-	-	-	-	-	-	1	
10	-	-	-	-	-	-	-	-	-	-	-	-	-	1	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TAB. 4.1 – Représentation d’une instance sous la forme d’une matrice. La largeur correspond au nombre de sommets de la chaîne (15) et le hauteur au nombre de requêtes (10). La première requête va du sommet 2 au sommet 10, on affecte donc les éléments de la matrice correspondant aux sommets émetteurs de la requête (sommets 2 à 9) avec la valeur 1. Note : les 0 ne sont pas affichés pour ne pas surcharger la vue.

Contraintes :

- $C_1 = \text{matriceDates}[i][j] < \text{matriceDates}[i][j + 1]$ pour tout (i, j) tels que $\text{matriceReq}[i][j] = \text{matriceReq}[i][j + 1] = 1$
- $C_2 =$ Dans *matriceDates*, une valeur dans une colonne doit être différente de toutes les valeurs des deux colonnes précédentes et des deux colonnes suivantes (excepté pour les 0).
- $C_3 =$ Générer toutes les solutions et minimiser la valeur maximale.

Avec l’instance précédente, le programme nous renverra la matrice des dates 4.2.

1	-	1	5	6	7	8	9	11	-	-	-	-	-	-	
2	-	-	2	3	4	5	6	7	8	9	10	11	12	-	
3	-	-	-	-	1	2	3	4	5	6	7	8	9	-	
4	-	-	-	-	-	-	-	10	-	-	-	-	-	-	
5	-	-	-	-	-	-	-	-	1	3	4	5	6	-	
6	-	-	-	-	-	-	-	-	2	12	13	-	-	-	
7	-	-	-	-	-	-	-	-	-	-	-	1	-	-	
8	-	-	-	-	-	-	-	-	-	-	-	-	2	-	
9	-	-	-	-	-	-	-	-	-	-	-	-	-	3	
10	-	-	-	-	-	-	-	-	-	-	-	-	-	4	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TAB. 4.2 – Affectation de dates optimale obtenue grâce au solveur CHOCO.

4.1.2 Analyse de l'algorithme

4.2 Performances des algorithmes

Pour valider les performances des algorithmes précédents nous les avons comparé avec la solution optimale sur plusieurs types d'instances. Le protocole est le suivant : on lance un script qui génère aléatoirement le nombre d'instances souhaitées (différents générateurs existent, pour générer un certain nombre de requêtes par instance, ou pour borner la charge par exemple). Pour chaque instance, nous lançons l'algorithme GD, l'algorithme DG, ainsi que le programme CHOCO. On garde les résultats dans un fichier pour nous permettre de comparer les performances des deux algorithmes en fonction de la solution optimale.

Notation :

- **Moyenne GD** : rapport d'approximation moyen obtenu grâce à l'algorithme GD par rapport à la solution optimale pour toutes les instances testées.
- **Moyenne DG** : rapport d'approximation moyen obtenu grâce à l'algorithme DG par rapport à la solution optimale pour toutes les instances testées.
- **Moyenne meilleur** : donne le rapport d'approximation moyen en choisissant le meilleur algorithme d'approximation pour chaque instance.
- **Optimal GD** : nombre d'instances pour lesquelles l'algorithme GD a renvoyé la solution optimale.
- **Optimal DG** : nombre d'instances pour lesquelles l'algorithme DG a renvoyé la solution optimale.
- **Pire GD** : pire rapport d'approximation à l'optimal de l'algorithme DG sur toutes les instances testées.
- **Pire DG** : pire rapport d'approximation à l'optimal de l'algorithme GD sur toutes les instances testées.

1 - 50 instances avec 10 sommets et 10 requêtes

Moyenne GD	Moyenne DG	Moyenne meilleur	
1,1	1,06	1,06	

Optimal GD	Optimal DG	Pire GD	Pire DG
13	23	1,5	1,3

Pour cette série d'instances, l'algorithme DG a été 25 fois meilleur que GD, qui de son côté a été meilleur 12 fois seulement. Les deux algorithmes ont renvoyés le même résultat 13 fois. Comme on peut le constater sur le tableau, pour des « petites » instances, les performances sont très bonnes, notamment pour l'algorithme DG qui retourne la solution optimale près d'une fois sur deux et a un rapport d'approximation moyen excellent (1,06).

2 - 50 instances avec 20 sommets et 20 requêtes

Moyenne GD	Moyenne DG	Moyenne meilleur	
1,17	1,09	1,08	

Optimal GD	Optimal DG	Pire GD	Pire DG
0	12	1,5	1,31

Pour des instances plus grosses, les performances moyennes sont très bonnes également (1,08) mais la découverte d'une solution optimale par les algorithmes d'approximation est plus rare (12 fois, par l'algorithme DG uniquement).

3 - 50 instances avec 20 sommets et une charge maximale égale à 8

Moyenne	Pire	Optimal
1,13	1,41	7

Dorénavant, on applique l'algorithme GD ainsi que l'algorithme DG sur chacune des instance et on ne garde que celui qui renvoie la meilleure solution des deux. Sur une chaîne de vingt sommets et avec une charge relativement importante, on s'aperçoit que les performances se dégradent. La solution optimale n'est atteinte que 7 fois et le rapport d'approximation moyen a augmenté, mais correct par rapport à la borne supérieure trouvée en 3.4.

4 - 50 instances avec 30 sommets et une charge maximale égale à 5

Moyenne	Pire	Optimal
1,02	1,16	38

Les algorithmes sont très efficaces si la charge est peu importante et répartie sur un nombre important de sommets puisque l'on atteint 38 fois la solution optimale, et le rapport d'approximation moyen est très bas.

Conclusion : Les résultats ci-dessus sont donc encourageants. Il sont particulièrement efficaces en moyenne et sur plus de 500 instances testées (qui ne figurent pas toutes dans ce rapport) aucune ne dépasse le rapport 1,5.

Chapitre 5

Conclusion et perspectives

Durant ce stage, nous avons étudié un problème algorithmique posé par les réseaux radio. Ce problème est NP-difficile et non-approximable dans le cas général. Nous avons d'abord étudié l'impact de la topologie du réseau en cherchant la complexité du problème dans les arbres avec la charge bornée à 1, et les chaînes. Nous avons ensuite proposé des algorithmes d'approximation de rapport 6 pour le problème sur la chaîne dont nous avons comparé les performances par rapport à la solution optimale, obtenue grâce à la programmation par contraintes et le solveur CHOCO. Ces résultats sont encourageants puisque sur les instances testées (plus de 500), le rapport n'est jamais supérieur à 1,5 par rapport à l'optimal.

De nombreuses questions restent encore sans réponse. Le principal résultat que nous attendons serait d'établir la complexité de DAWN-paths dans les chaînes. Un résultat de NP-Complétude donnerait un sens plus profond à notre algorithme d'approximation. Un résultat de polynomialité permettrait d'inclure une nouvelle famille de graphes sur laquelle on peut espérer résoudre de grandes instances. Actuellement, DAWN-paths n'est polynomial que sur les étoiles, et est NP-Complet sur de nombreuses classes de graphes, même très restrictives comme les arbres. Un tel résultat ouvrirait de nouvelles perspectives pour établir la complexité de DAWN-paths dans d'autres topologies, notamment les arbres enracinés ou les grilles. C'est dans ce sens que nous avons proposé une nouvelle heuristique que nous avons montré plus efficace en pratique. Il est possible que cette heuristique puisse être améliorée afin de devenir un algorithme exact. Pour plus de détails, le lecteur se reportera à la section A.2 en annexe de ce rapport. Le groupe de travail se penche actuellement sur une preuve lorsque les requêtes sont toutes orientées dans le même sens.

L'étude du problème sur la chaîne lorsque la charge est bornée est également une perspective envisageable. Soulignons enfin que le facteur d'approximation de nos heuristiques a été calculée grossièrement, et qu'une étude plus approfondie permettrait de tirer un meilleur rapport.

Bibliographie

- [1] G. Chelius. *Architectures et Communications dans les réseaux spontanés sans fil*. PhD thesis, INSA de Lyon, INRIA Rhône Alpes, France, April 2004.
- [2] B. Darties and J. Palaysi. Satisfaction de requêtes par affectation de dates d'émissions dans les réseaux radios. In *Rencontres francophones du Parallélisme (RenPar'17)*, 2006.
- [3] I. Chlamtac and S. Kutten. On broadcasting in radio networks - problem analysis and protocol design. *IEEE Transactions on Communications*, 33 :1240–1246, December 1985.
- [4] J.-C. Bermond and J. Peters. Efficient gathering in radio grids with interference. In *AlgoTel'05*, Presqu'île de Giens, May 2005.
- [5] J.-C. Bermond, J. Galtier, R. Klasing, N. Morales, and S. Pérennes. Hardness and approximation of gathering in static radio networks. In *FAWN06, Pisa, Italy*, March 2006.
- [6] Olivier Cogis et Claudine Robert. *Théorie des graphes : Au-delà des ponts de Königsberg, Problèmes, théorèmes, algorithmes*. Vuibert, 2003.
- [7] J.K. Lenstra R.L Graham, E.L. Lawler and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Ann. Discrete Math.*, 5 :287–326, 1979.
- [8] E. Pesch J. Blazewicz, K. H. Ecker. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag Berlin, 1996.
- [9] Sylvain Durand B. Darties and J. Palaysi. Satisfaction de requêtes dans un réseau radio synchrone : Np-complétude dans les arbres. 2007.
- [10] <http://choco-solver.net>.

Annexe A

Annexes

A.1 Efficacité des algorithmes

Exemple d'instance pour laquelle l'algorithme DG renvoie une meilleure solution que l'algorithme GD. Sur une chaîne de 10 sommets nous avons une collection de 5 requêtes telles que $r_1 = (1, 6)$, $r_2 = (1, 3)$, $r_3 = (2, 5)$, $r_4 = (3, 10)$, $r_5 = (9, 10)$.

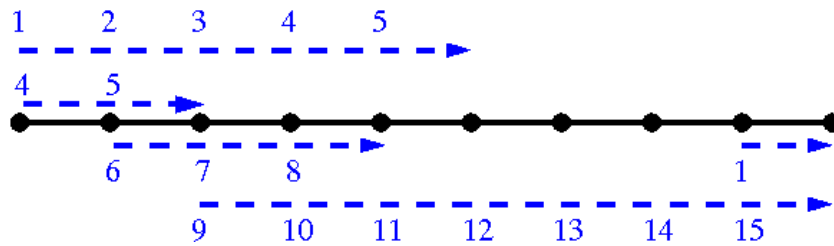


FIG. A.1 – Avec l'algorithme GD, la date maximale assignée est 15.

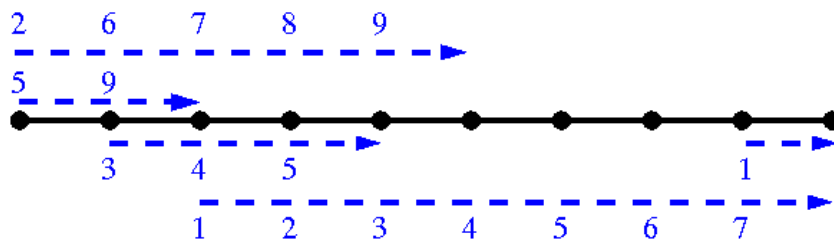


FIG. A.2 – Avec l'algorithme DG, la date maximale assignée est 9.

Exemple d'instance pour laquelle l'algorithme GD renvoie une meilleure solution que l'algorithme DG. Sur une chaîne de 10 sommets nous avons une collection de 5 requêtes telles que $r_1 = (2, 7)$, $r_2 = (3, 4)$, $r_3 = (4, 6)$, $r_4 = (6, 8)$, $r_5 = (7, 8)$.

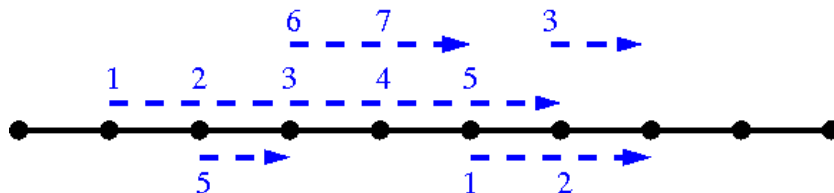


FIG. A.3 – Avec l'algorithme GD, la date maximale assignée est 7.

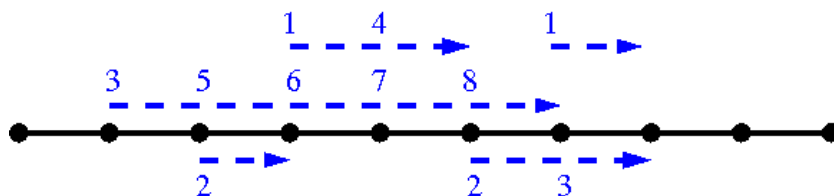


FIG. A.4 – Avec l'algorithme DG, la date maximale assignée est 8.

A.2 Encore un algorithme sur la chaîne

Cet algorithme, détaillé ci-dessous, fonctionne par étape. À l'étape i , on sélectionne tous les sommets du réseaux qui émettrons à la date i . Pour choisir quels seront ces sommets, on procède à une comparaison des requêtes dont ils sont sur le parcours. À l'heure actuelle nous ne pouvons pas garantir la polynomialité de cet algorithme.

A.2.1 Algorithme

Algorithme 3 : fonction $meilleur(r_1, r_2)$

Entrées : Deux requêtes r_1, r_2

Sorties : La requête à privilégier

Cette fonction compare deux requêtes, et renvoie celle qui est à privilégier pour être affectée à l'étape actuelle.

Algorithme 4 : Nouvel algorithme à tester.

Entrées : Une instance DAWN-paths sur la chaîne
Sorties : Une assignation de dates f correcte et sans conflit
Ordre O ;
Ensemble $SELECT$;
Ordre O_{next} ;
requête $rcourant$;
requête $rpile$;
requête $rlast$;
Pile P ;
Int $etape$;
début
 $etape=1$
 $O=$ Ordre sur les requêtes : r_i est avant r_j ssi $s(r_i)$ est à droite de $s(r_j)$
 tant que $etape \leq D$ et $O \neq \emptyset$ **faire**
 tant que $O \neq \emptyset$ **faire**
 $rcourant = \text{extrairePremiereRequete}(O)$
 si $rcourant$ n'est pas en conflit avec l'élément suivant dans O
 ou O est vide **alors**
 | ajouter $rcourant$ à $SELECT$
 sinon
 tant que $rcourant$ est en conflit avec la première requête
 de O et O n'est pas vide **faire**
 $rdefi = \text{extrairePremiereRequete}(O)$
 si $\text{meilleur}(rcourant, rdefi) = rcourant$ **alors**
 | empiler($rdefi, P$)
 sinon
 | empiler($rcourant, P$)
 | $rcourant = rdefi$
 ajouter $rcourant$ à $SELECT$
 $rlast = rcourant$
 tant que $P \neq \emptyset$ **faire**
 $rpile = \text{depiler}(P)$
 si $rpile$ est en conflit avec $rlast$ **alors**
 | ajouter $rpile$ à O_{next}
 sinon
 | ajouter $rpile$ à $SELECT$
 | $rlast = rpile$
 Faire émettre, puis supprimer le premier sommet de chaque
 élément de $SELECT$ à la date $etape$
 Ajouter les éléments de $SELECT$ dans O_{next}
 trier O_{next}
 $O = O_{next}$
 vider $SELECT$
 vider O_{next}
 $etape++$
fin
